# UNIX System

# Activity Package

This document is part of the ADMINISTRATOR'S GUIDE. Therefore the pagenumbers don't begin with 1.

## 11. UNIX SYSTEM ACTIVITY PACKAGE

### General

This section describes the design and implementation of the UNIX System Activity Package. The UNIX operating system contains a number of counters that are incremented as various system actions occur. The system activity package reports UNIX system-wide measurements including Central Processing Unit(CPU) utilization, disk and tape Input/Output(I/O) activities, terminal device activity, buffer usage, system calls, system switching and swapping, file-access activity, queue activity, and message and semaphore activities. The package provides four commands that generate various types of reports. Procedures that automatically generate daily reports are also included. The five functions of the activity package are:

- **sar**(1) command—allows a user to generate system activity reports in real-time and to save system activities in a file for later usage.
- **sag**(1G) command—displays system activity in a graphical form.
- **sadp**(1) command—samples disk activity once every second during a specified time interval and reports disk usage and seek distance in either tabular or histogram form.
- **timex**(1)—a modified **time**(1) command that times a command and also reports concurrent system activity.
- system activity daily reports—procedures are provided for sampling and saving system activities in a data file periodically and for generating the daily report from the data file.

The system activity information reported by this package is derived from a set of system counters located in the operation system kernel. These system counters are described in the part "System Activity Counters". The part "System Activity Commands" describes the commands provided by this package. The procedure for generating daily reports is given in "Daily Report Generation". A description for each of the files used by the system activity package can be found in Attachment 11.1.

### System Activity Counters

The UNIX operating system manages a number of counters that record various activities and provide the basis for the system activity reporting system. The data structure for most of these counters is defined in the *sysinfo* structure (see Attachment 11.2) in */usr/include/sys/sysinfo.h*. The system table overflow counters are kept in the *_syserr* structure. The device activity counters are extracted from the device status tables. In this version, the I/O activity of the following devices is recorded: RP06, RM05, RS04, RF11, RK05, RP03, RL02, TM03, and TM11.

In the following paragraphs, the system activity counters that are sampled by the system activity package are described.

***Cpu time counters:*** There are four time counters that may be incremented at each clock interrupt 60 times per second. Exactly one of the *cpu*[] counters is incremented on each interrupt, according to the mode the CPU is in at the interrupt; idle, user, kernal, and wait for I/O completion.

***Lread and lwrite:*** The *lread* and *lwrite* counters are used to count logical read and write requests issued by the system to block devices.

***Bread and bwrite:*** The *bread* and *bwrite* counters are used to count the number of times data is transferred between the system buffers and the block devices. These actual I/Os are triggered by logical I/Os that

cannot be satisfied by the current contents of the buffers. The ratio of block I/O to logical I/O is a common measure of the effectiveness of the system buffering.

***Phread and phwrite:*** The *phread* and *phwrite* counters count read and write requests issued by the system to raw devices.

***Swapin and swapout:*** The *swapin* and *swapout* counters are incremented for each system request initiating a transfer from or to the swap device. More than one request is usually involved in bringing a process into memory, or out, because text and data are handled separately. Frequently used programs are kept on the swap device and are swapped in rather than loaded from the file system. The *swapin* counter reflects these initial loading operations as well as resumptions of activity, while the *swapout* counter reveals the level of actual "swapping." The amount of data transferred between the swap device and memory are measured in blocks and counted by *bswapin* and *bswapout*.

***Pswitch and syscall:*** These counters are related to the management of multiprogramming. *Syscall* is incremented every time a system call is invoked. The numbers of invocations of **read**(2), **write**(2), **fork**(2), and **exec**(2) system calls are kept in counters *sysread, syswrite, sysfork,* and *sysexec,* respectively. *Pswitch* counts the times the switcher was invoked, which occurs when:

a. a system call resulted in a road block

b. an interrupt occurred resulting in awakening a higher priority process

c. 1-second clock interrupt.

***Iget, namei, and dirblk:*** These counters apply to file-access operations. *Iget* and *namei,* in particular, are the names of UNIX operating system routines. The counters record the number of times that the respective routines are called. *Namei* is the routine that performs file system path searches. It searches the various directory files to get the associated i-number of a file corresponding to a special path. *Iget* is a routine called to locate the inode entry of a file (i-number). It first searches the in-core inode table. If the inode entry is not in the table, routine *iget* will get the inode from the file system where the file resides and make an entry in the in-core inode table for the file. *Iget* returns a pointer to this entry. *Namei* calls *iget,* but other file access routines also call *iget*. Therefore, counter *iget* is always greater than counter *namei*.

Counter *dirblk* records the number of directory block reads issued by the system. It is noted that the directory blocks read divided by the number of *namei* calls estimates the average path length of files.

***Runque, runocc, swpque, and swpocc:*** These counters are used to record queue activities. They are implemented in the *clock.c* routine. At every 1 second interval, the clock routine examines the process table to see whether any processes are in core and in ready state. If so, the counter *runocc* is incremented and the number of such processes are added to counter *runque*. While examining the process table, the clock routine also checks whether any processes in the swap device are in ready state. The counter *swpocc* is incremented if the swap queue is occupied, and the number of processes in swap queue is added to counter *swpque*.

***Readch and writech:*** The *readch* and *writech* counters record the total number of bytes (characters) transferred by the **read** and **write** system calls, respectively.

***Monitoring terminal device activities:*** There are six counters monitoring terminal device activities. *Rcvint, xmtint,* and *mdmint* are counters measuring hardware interrupt occurrences for receiver, transmitter, and modem individually. *Rawch, canch,* and *outch* count number of characters in the raw queue, canonical queue, and output queue. Characters generated by devices operating in the *cooked* mode, such as terminals, are counted in both *rawch* and (as edited) in *canch,* but characters from raw devices, such as communication processors, are counted only in *rawch*.

***Msg and sema counters:*** These counters record message sending and receiving activities and semaphore operations, respectively.

***Monitoring I/O activities:*** As to the I/O activity for a disk or tape device, four counters are kept for each disk or tape drive in the device status table. Counter *io_ops* is incremented when an I/O operation has occurred on the device. It includes block I/O, swap I/O, and physical I/O. *Io_bcnt* counts the amount of data transferred between the device and memory in 512 byte units. *Io_act* and *io_resp* measure the active time and response time of a device in time ticks summed over all I/O requests that have completed for each device. The device active time includes the device seeking, rotating and data transferring times, while the response time of an I/O operation is from the time the I/O request is queued to the device to the time when the I/O completes.

***Inodeovf, fileovf, textovf, and procovf:*** These counters are extracted from *_syserr* structure. When an overflow occurs in any of the inode, file, text and process tables, the corresponding overflow counter is incremented.

#### System Activity Commands

The system activity package provides three commands for generating various system activity reports and one command for profiling disk activities. These tools facilitate observation of system activity during

- a controlled stand-alone test of a large system
- an uncontrolled run of a program to observe the operating environment
- normal production operation.

Commands **sar** and **sag** permit the user to specify a sampling interval and number of intervals for examining system activity and then to display the observed level of activity in tabular or graphical form. The **timex** command reports the amount of system activity that occurred during the precise period of execution of a timed command. The **sadp** command allows the user to establish a sampling period during which access location and seek distance on specified disks are recorded and later displayed as a tabular summary or as a histogram.

#### The "sar" command

The **sar** command can be used in the following ways:

- When the frequency arguments **t** and **n** are specified, it invokes the data collection program **sadc** to sample the system activity counters in the operating system every **t** seconds for **n** intervals and generates system activity reports in real-time. Generally, it is desirable to include the option to save the sampled data in a file for later examination. The format of the data file is shown in **sar**(8). In addition to the system counters, a time stamp is also included. It gives the time at which the sample was taken.
- If no frequency arguments are supplied, it generates system activity reports for a specified time interval from an existing data file that was created by **sar** at an earlier time.

A convenient usage is to run **sar** as a background process, saving its samples in a temporary file but sending its standard output to */dev/null*. Then an experiment is conducted after which the system activity is extracted from the temporary file. The **sar**(1) manual entry describes the usage and lists various types of reports. Attachment 11.3 gives formula for deriving each reported item.

#### The "sag" command

**Sag** displays system activity data graphically. It relies on the data file produced by a prior run of **sar** after which any column of data or the combination of columns of data of the **sar** report can be plotted. A fairly simple but powerful command syntax allows the specification of cross plots or time plots. Data items are selected using the **sar** column header names. The **sar**(1G) manual entry describes its options and usage. The system activity

graphical program invokes **graphics**(1G) and **tplot**(1G) commands to have the graphical output displayed on any of the terminal types supported by **tplot**.

#### The "timex" command

The **timex** command is an extension of the **time**(1) command. Without options, **timex** behaves exactly like **time**. In addition to giving the time information, it also prints a system activity report derived from the system counters. The manual entry **timex**(1) explains its usage. It should be emphasized that the user and sys times reported in the second and third lines are for the measured process itself including all its children while the remaining data (including the cpu user % and cpu sys %) are for the entire system.

While the normal use of **timex** will probably be to measure a single command, multiple commands can also be timed; either by combining them in an executable file and timing it, or more concisely, by typing:

```
timex sh -c " cmd1; cmd2; ... ; "
```

This establishes the necessary parent-child relationships to correctly extract the user and system times consumed by **cmd1**,**cmd2**, . . . (and the shell).

#### The "sadp" command

**Sadp** is a user level program that can be invoked independently by any user. It requires no storage or extra code in the operating system and allows the user to specify the disks to be monitored. The program is reawakened every second, reads system tables from */dev/kmem*, and extracts the required information. Because of the 1 second sampling, only a small fraction of disk requests are observed; however, comparative studies have shown that the statistical determination of disk locality is adequate when sufficient samples are collected.

In the operating system, there is an *iobuf* for each disk drive. It contains two pointers which are head and tail of the I/O active queue for the device. The actual requests in the queue may be found in three buffer header pools—system buffer headers for block I/O requests, physical buffer headers for physical I/O requests, and swap buffer headers for swap I/O. Each buffer header has a forward pointer which points to the next request in the I/O active queue and a backward pointer which points to the previous request.

**Sadp** snapshots the *iobuf* of the monitored device and the three buffer header pools once every second during the monitoring period. It then traces the requests in the I/O queue, records the disk access location, and seeks distance in buckets of 8 cylinder increments. At the end of monitoring period, it prints out the sampled data. The output of **sadp** can be used to balance load among disk drives and to rearrange the layout of a particular disk pack. The usage of this command is described in manual entry **sadp**(1).

### Daily Report Generation

The previous part described the commands available to users to initiate activity observations. It is probably desirable for each installation to routinely monitor and record system activity in a standard way for historical analysis. This part describes the steps that a system administrator may follow to automatically produce a standard daily report of system activity.

#### Facilities

- **sadc**—The executable module of **sadc.c** (see Attachment 11.1) which reads system counters from */dev/kmem* and records them to a file. In addition to the file argument, two frequency arguments are usually specified to indicate the sampling interval and number of samples to be taken. In case no frequency arguments are given, it writes a dummy record in the file to indicate a system restart.

- **sa1**—The shell procedure that invokes **sadc** to write system counters in the daily data file */usr/adm/sa* **dd** where **dd** represents the day of the month. It may be invoked with sampling interval and iterations as arguments.

- **sa2**—The shell procedure that invokes the **sar** command to generate daily report */usr/adm/sa/sar***dd** from the daily data file */usr/adm/sa/sa***dd**. It also removes daily data files and report files after 7 days. The starting and ending times and all report options of **sar** are applicable to **sa2**.

**Suggested Operational Setup**

It is suggested that the **cron**(1M) control the normal data collection and report generation operations. For example, the sample entries in */usr/lib/crontab:*

```
0 * * * 0,6 su sys -c " /usr/lib/sa/sa1 "
0 18- * * 1-5 su sys -c " /usr/lib/sa/sa1 "
0 8-17 * * 1-5 su sys -c " /usr/lib/sa/sa1 1200 3 "
```

would cause the data collection program **sadc** to be invoked every hour on the hour. Moreover, depending on the arguments presented, it writes data to the data file one to three times at every 20 minutes. Therefore, under the control of **cron**(1M), the data file is written every 20 minutes between 8:00 and 18:00 on weekdays and hourly at other times.

Note that data samples are taken more frequently during prime time on weekdays to make them available for a finer and more detailed graphical display. It is suggested that **sa1** be invoked hourly rather than invoking it once every day; this ensures that if the system crashes data collection will be resumed within an hour after the system is restarted.

Because system activity counters restart from zero when the system is restarted, a special record is written on the data file to reflect this situation. This process is accomplished by invoking **sadc** with no frequency arguments within */etc/rc* when going to multiuser state:

```
su adm -c " /usr/lib/sa/sadc /usr/adm/sa/sa'date +%d' "
```

**Cron**(1M) also controls the invocation of **sar** to generate the daily report via shell procedure **sa2**. One may choose the time period the daily report is to cover and the groups of system activity to be reported. For instance, if:

```
0 20 * * 1-5 su sys -c " /usr/lib/sa/sa2 -s 8:00 -e 18:00 -i 3600 -uybd "
```

is an entry in */usr/lib/crontab*, **cron** will execute the **sar** command to generate daily reports from the daily data file at 20:00 on weekdays. The daily report reports the CPU utilization, terminal device activity, buffer usage, and device activity every hour from 8:00 to 18:00.

In case of a shortage of the disk space or for any other reason, these data files and report files can be removed by the superuser. The manual entry **sar**(8) describes the daily report generation procedure.

## ATTACHMENT 11.1

The source files and shell programs of the system activity package are in directory */usr/src/cmd/sa*.

| | |
|---|---|
| **sa.h** | The system activity header file defines the structure of data file and device information for measured devices. It is included in **sadc.c**, **sar.c**, and **timex.c**. |
| **sadc.c** | The data collection program that accesses */dev/kmem* to read the system activity counters and writes data either on standard output or on a binary data file. It is invoked by the **sar** command generating a real-time report. It is also invoked indirectly by entries in */usr/lib/crontab* to collect system activity data. |
| **sar.c** | The report generation program invokes **sadc** to examine system activity data, generates reports in real-time, and saves the data to a file for later usage. It may also generate system activity reports from an existing data file. It is invoked indirectly by **cron** to generate daily reports. |
| **saghdr.h** | The header file for **saga.c** and **sagb.c**. It contains data structures and variables used by **saga.c** and **sagb.c**. |
| **saga.c & sagb.c** | The graph generation program that first invokes **sar** to format the data of a data file in a tabular form and then displays the **sar** data in graphical form. |
| **sa1.sh** | The shell procedure that invokes **sadc** to write data file records. It is activated by entries in */usr/lib/crontab*. |
| **sa2.sh** | The shell procedure that invokes **sar** to generate the report. It also removes the daily data files and daily report files after a week. It is activated by an entry in */usr/lib/crontab* on weekdays. |
| **timex.c** | The program that times a command and generates a system activity report. |
| **sadp.c** | The program that samples and reports disk activities. |

# LP Spooling System

This document is part of the ADMINISTRATOR'S GUIDE. Therefore the pagenumbers don't begin with 1.

# 9. LP SPOOLING SYSTEM

## GENERAL

The LP program is a system of commands which performs diverse spooling functions under the UNIX operating system. Since the primary LP application is off-line printing, this document focuses mainly on spooling to line printers. LP allows administrators to customize the system to spool to a collection of line printers of any type and to group printers into logical classes in order to maximize the throughput of the devices. Users are provided the capabilities of queuing and canceling print requests, preventing and allowing queuing to and printing on devices, starting and stopping LP from processing requests, changing configuration of printers and finding status of the LP system. This section describes the role of an LP Administrator in performing restricted functions and overseeing the smooth operation of LP.

Throughout this section, each reference of the form **name**(1M), **name**(7), or **name**(8) refers to entries in the UNIX System Administrator's Manual. All other references to entries of the form **name**(N), where "N" is a number (1 through 6) possibly followed by a letter, refers to entry **name** in Section "N" of the UNIX System User's Manual.

## OVERVIEW OF LP FEATURES

### A. Definitions

Several terms must be defined before presenting a brief summary of LP commands. The LP was designed with the flexibility to meet the needs of users on different UNIX systems. Changes to the LP configuration are performed by the **lpadmin**(1M) command.

LP makes a distinction between printers and printing devices. A *device* is a physical peripheral device or a file and is represented by a full UNIX system pathname. A *printer* is a logical name that represents a device. At different points in time, a printer may be associated with different devices. A *class* is a name given to an ordered list of printers. Every class must contain at least one printer. Each printer may be a member of zero or more classes. A *destination* is a printer or a class. One destination may be designated as the *system default destination*. The **lp**(1) command will direct all output to this destination unless the user specifies otherwise. Output that is routed to a printer will be printed only by that printer, whereas output directed to a class will be printed by the first available class member.

Each invocation of **lp** creates an output request that consists of the files to be printed and options from the lp command line. An interface program which formats requests must be supplied for each printer. The LP scheduler, **lpsched**(1M), services requests for all destinations by routing requests to interface programs to do the printing on devices. An LP configuration for a system consists of devices, destinations, and interface programs.

### B. Commands

#### Commands for General Use

The **lp**(1) command is used to request the printing of files. It creates an output request and returns a request id of the form

dest-seqno

to the user, where *seqno* is a unique sequence number across the entire LP system, and *dest* is the destination where the request was routed.

**Cancel** is used to cancel output requests. The user supplies request ids as returned by **lp** or printer names, in which case the currently printing requests on those printers are canceled.

**Disable** prevents **lpsched** from routing output requests to printers.

**Enable**(1) allows **lpsched** to route output requests to printers.

#### Commands for LP Administrators

Each LP system must designate a person or persons as LP administrator to perform the restricted functions listed below. Either the superuser or any user who is logged into the UNIX system as **lp** qualifies as an LP Administrator. All LP files and commands are owned by **lp**, except for **lpadmin** and **lpsched** which are owned by root. The following commands will be described in more detail later in this section.

| | |
|---|---|
| **Lpadmin**(1M) | Modifies LP configuration. Many features of this command cannot be used when **lpsched** is running. |
| **Lpsched**(1M) | Routes output requests to interface programs which do the printing on devices. |
| **Lpshut** | Stops **lpsched** from running. All printing activity is halted, but other LP commands may still be used. |
| **Accept**(1M) | Allows **lp** to accept output requests for destinations. |
| **Reject** | Prevents **lp** from accepting requests for destinations. |
| **Lpmove** | Moves output requests from one destination to another. Whole destinations may be moved at once. This command cannot be used when **lpsched** is running. |

### BUILDING LP

All LP commands are built from source code that resides in the */usr/src/cmd/lp* directory including the make file, *lp.mk*. Unless some of the definitions in *lp.mk* are changed, LP may be installed only by the superuser. Before installing a new LP system, make sure there is a login called *lp* on your system and that the spool directory, */usr/spool/lp*, does not exist. To install LP, perform the following:

```
cd /usr/src/cmd/lp
make -f lp.mk install
```

This builds all LP commands and creates an initial LP configuration consisting of no printers, classes, or default destination. LP must be configured by an LP administrator using the **lpadmin** command in order to create a useful spooler.

In addition, add the following code to */etc/rc*:

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
echo " LP scheduler started "
```

This starts the LP scheduler each time that the UNIX system is restarted.

Several variables in *lp.mk* may be changed before installing LP to customize the system:

| Variable | Default Value | Meaning |
|---|---|---|
| SPOOL | */usr/spool/lp* | spool directory |
| ADMIN | *lp* | logname of LP Administrator |

| | | |
|---|---|---|
| GROUP | *bin* | group owning LP commands/data |
| ADMDIR | */usr/lib* | commands of administrator |
| USRDIR | */usr/bin* | user commands reside here |

If an existing LP spool directory is corrupted (but not the LP programs) or if it needs to be rebuilt from scratch, make sure that **lpsched** is not running and perform the following as superuser:

1. Make copies of any interface programs that are not standard LP software. **DO NOT** make these copies underneath the spool directory. The pathname for printer "p" is */usr/spool/lp/interface/p*.
2. rm −fr /usr/spool/lp
3. Make *−f lp.mk* new. (This recreates the bare LP configuration described above.)

PRECAUTIONS:

1. Some LP commands invoke other LP commands. Moving them after they are built will cause some commands to fail.
2. The files under the SPOOL directory should be modified **only by LP commands**.
3. All LP commands require set-user-id permission. If this is removed, the commands will fail.

### CONFIGURING LP—THE "lpadmin" COMMAND

Changes to the LP configuration should be made by using the **lpadmin** command and not by hand. **Lpadmin** will not attempt to alter the LP configuration when **lpsched** is running, except where explicitly noted below.

#### A. Introducing New Destinations

The following information must be supplied to **lpadmin** when introducing a new printer:

1. The printer name (**−p** printer) is an arbitrary name which must conform to the following rules:
   - It must be no longer than 14 characters.
   - It must consist solely of alphanumeric characters and underscores.
   - It must not be the name of an existing LP destination (printer or class).
2. The device associated with the printer (**−v** device). This is the pathname of a hardwired printer, a login terminal, or other file that is writable by lp.
3. The printer interface program. This may be specified in one of three ways:
   - It may be selected from a list of model interfaces supplied with LP (**−m** model).
   - It may be the same interface that an existing printer uses (**−e** printer).
   - It may be a program supplied by the LP administrator (**−i** interface).

Information which need not always be supplied when creating a new printer includes:

1. The user may specify **−h** to indicate that the device for the printer is hardwired or the device is the name of a file (this is assumed by default). If, on the other hand, the device is the pathname of a login terminal,

then −l must be included on the command line. This indicates to *lpsched* that it must automatically disable this printer each time *lpsched* starts running. This fact is reported by *lpstat* when it indicates printer status:

```
$ lpstat −pa
printer a (login terminal) disabled Oct 31 11:15−
        disabled by scheduler: login terminal
```

This is done because device names for login terminals can be (and usually are) associated with different physical devices from day to day. If the scheduler did not take this action, somebody might log in and be surprised that LP is spooling to his/her terminal!

2. The new printer may be added to an existing class or added to a new class (−cclass). New class names must conform to the same rules for new printer names.

## EXAMPLES

The following examples will be referenced by further examples in later sections.

1. Create a printer called pr1 whose device is /dev/printer and whose interface program is the model hp interface:

```
$ /usr/lib/lpadmin −ppr1 −v/dev/printer −mhp
```

2. Add a printer called pr2 whose device is /dev/tty22 and whose interface is a variation of the model prx interface. It is also a login terminal:

```
$ cp /usr/spool/lp/model/prx xxx
   < edit xxx >
$ /usr/lib/lpadmin −ppr2 −v/dev/tty22 −ixxx −l
```

3. Create a printer called pr3 whose device is /dev/tty23. The pr3 will be added to a new class called cl1 and will use the same interface as printer pr2:

```
$ /usr/lib/lpadmin −ppr3 −v/dev/tty23 −epr2 −ccl1
```

### B. Modifying Existing Destinations

Modifications to existing destinations must always be made with respect to a printer name (−**p** printer). The modifications may be one or more of the following:

1. The device for the printer may be changed (−**v** device). If this is the only modification, then this may be done even while *lpsched* is running. This facilitates changing devices for login terminals.

2. The printer interface program may be changed (−**m** model, −**e** printer, −**i** interface).

3. The printer may be specified as hardwired (−**h** ) or as a login terminal (−**l** ).

4. The printer may be added to a new or existing class (−cclass).

5. The printer may be removed from an existing class (−**r** class). Removing the last remaining member of a class causes the class to be deleted. No destination may be removed if it has pending requests. In that case, **lpmove** or **cancel** should be used to move or delete the pending requests.

EXAMPLES

These examples are based on the LP configuration created by those in the previous section.

1. Add printer pr2 to class cl1:

```
$ /usr/lib/lpadmin -ppr2 -ccl1
```

2. Change pr2's interface program to the model prx interface, change its device to /dev/tty24, and add it to a new class called cl2:

```
$ /usr/lib/lpadmin -ppr2 -mprx -v/dev/tty24 -ccl2
```

Note that printers pr2 and pr3 now use different interface programs even though pr3 was originally created with the same interface as pr2. Printer pr2 is now a member of two classes.

3. Specify printer pr2 as a hardwired printer:

```
$ /usr/lib/lpadmin -ppr2 -h
```

4. Add printer pr1 to class cl2:

```
$ /usr/lib/lpadmin -ppr1 -ccl2
```

The members of class cl2 are now pr2 and pr1, in that order. Requests routed to class cl2 will be serviced by pr2 if both pr2 and pr1 are ready to print; otherwise, they will be printed by the one which is next ready to print.

5. Remove printers pr2 and pr3 from class cl1:

```
$ /usr/lib/lpadmin -ppr2 -rcl1
$ /usr/lib/lpadmin -ppr3 -rcl1
```

Since pr3 was the last remaining member of class cl1, the class is removed.

6. Add pr3 to a new class called cl3.

```
$ /usr/lib/lpadmin -ppr3 -ccl3
```

### C. Specifying the System Default Destination

The system default destination may be changed even when **lpsched** is running.

EXAMPLES

1. Establish class cl1 as the system default destination:

```
$ /usr/lib/lpadmin -dcl1
```

2. Establish no default destination:

```
$ /usr/lib/lpadmin -d
```

### D. Removing Destinations

Classes and printers may be removed only if there are no pending requests that were routed to them. Pending requests must either be canceled using **cancel** or moved to other destinations using **lpmove** before destinations may be removed. If the removed destination is the system default destination, then the system will have

no default destination until the default destination is respecified. When the last remaining member of a class is removed, then the class is also removed. The removal of a class never implies the removal of printers.

### EXAMPLES

1. Make printer pr1 the system default destination:

```
$ /usr/lib/lpadmin -dpr1
```

   Remove printer pr1:

```
$ /usr/lib/lpadmin -xpr1
```

   Now there is no system default destination.

2. Remove printer pr2:

```
$ /usr/lib/lpadmin -xpr2
```

   Class cl2 is also removed since pr2 was its only member.

3. Remove class cl3:

```
$ /usr/lib/lpadmin -xcl3
```

   Class cl3 is removed, but printer pr3 remains.

## MAKING AN OUTPUT REQUEST—THE "lp" COMMAND

Once LP destinations have been created, users may request output by using the **lp** command. The request id that is returned may be used to see if the request has been printed or to cancel the request.

The LP program determines the destination of a request by checking the following list in order:

- If the user specifies *-d* dest on the command line, then the request is routed to *dest*.
- If the environment variable **LPDEST** is set, the request is routed to the value of *LPDEST*.
- If there is a system default destination, then the request is routed there.
- Otherwise, the request is rejected.

### EXAMPLES

1. There are at least four ways to print the password file on the system default destination:

```
lp /etc/passwd
lp < /etc/passwd
cat /etc/passwd | lp
lp -c /etc/passwd
```

   The last three ways cause copies of the file to be printed, whereas the first way prints the file directly. Thus, if the file is modified between the time the request is made and the time it is actually printed, then the changes will be reflected in the output.

2. Print two copies of file abc on printer xyz and title the output "my file":

```
pr abc | lp -dxyz -n2 -t " my file "
```

3. Print file xxx on a Diablo* 1640 printer called zoo in 12-pitch and write to the user's terminal when printing has completed:

```
lp -dzoo -o12 -w xxx
```

In this example, "12" is an option that is meaningful to the model Diablo 1640 interface program that prints output in 12-pitch mode [see **lpadmin**(1M)].

### FINDING LP STATUS—LPSTAT

The **lpstat** command is used to find status information about LP requests, destinations, and the scheduler.

#### EXAMPLES

1. List the status of all pending output requests made by this user:

```
lpstat
```

The status information for a request includes the request id, the logname of the user, the total number of characters to be printed, and the date and time the request was made.

2. List the status of printers p1 and p2:

```
lpstat -pp1,p2
```

### CANCELING REQUESTS—CANCEL

The LP requests may be canceled using the **cancel** command. Two kinds of arguments may be given to the command—request ids and printer names. The requests named by the request ids are canceled and requests that are currently printing on the named printers are canceled. Both types of arguments may be intermixed.

#### EXAMPLE

Cancel the request that is now printing on printer xyz:

```
cancel xyz
```

If the user that is canceling a request is not the same one that made the request, then mail is sent to the owner of the request. LP allows any user to cancel requests in order to eliminate the need for users to find LP administrators when unusual output should be purged from printers.

### ALLOWING AND REFUSING REQUESTS—ACCEPT AND REJECT

When a new destination is created, **lp** will reject requests that are routed to it. When the LP administrator is sure that it is set up correctly, he or she should allow **lp** to accept requests for that destination. The **accept** command performs this function.

Sometimes it is necessary to prevent **lp** from routing requests to destinations. If printers have been removed or are waiting to be repaired or if too many requests are building for printers, then it may be desirable to cause

* Trademark of Diablo Systems, Inc.

lp to reject requests for those destinations. The **reject** command performs this function. After the condition that led to the rejection of requests has been remedied, the **accept** command should be used to allow requests to be taken again.

The acceptance status of destinations is reported by the **−a** option of **lpstat.**

### EXAMPLES

1. Cause **lp** to reject requests for destination xyz:

```
/usr/lib/reject −r " printer xyz needs repair "  xyz
```

Any users that try to route requests to xyz will encounter the following:

```
$ lp −dxyz file
lp: can not accept requests for destination " xyz "
        −printer xyz needs repair
```

2. Allow **lp** to accept requests routed to destination xyz:

```
/usr/lib/accept xyz
```

#### ALLOWING AND INHIBITING PRINTING—ENABLE AND DISABLE

The **enable** command allows the LP scheduler to print requests on printers. That is, the scheduler routes requests only to the interface programs of enabled printers. Note that it is possible to enable a printer but to prevent further requests from being routed to it.

The **disable** command cancels the effects of the **enable** command. It prevents the scheduler from routing requests to printers, independently of whether or not **lp** is allowing them to accept requests. Printers may be disabled for several reasons including malfunctioning hardware, paper jams, and end of day shutdowns. If a printer is busy at the time it is disabled, then the request that it was printing will be reprinted in its entirety either on another printer (if the request was originally routed to a class of printers) or on the same one when the printer is reenabled. The **−c** option causes the currently printing requests on busy printers to be canceled in addition to disabling the printers. This is useful if strange output is causing a printer to behave abnormally.

### EXAMPLE

Disable printer xyz because of a paper jam:

```
$ disable −r " paper jam "  xyz
printer " xyz "  now disabled
```

Find the status of printer xyz:

```
$ lpstat −pxyz
printer " xyz "  disabled since Jan 5 10:15 —
          paper jam
```

Now, reenable xyz:

```
$ enable xyz
printer " xyz "  now enabled
```

### MOVING REQUESTS BETWEEN DESTINATIONS—LPMOVE

Occasionally, it is useful for LP administrators to move output requests between destinations. For instance, when a printer is down for repairs, it may be desirable to move all of its pending requests to a working printer. This is one way to use the **lpmove** command. The other use of this command is to move specific requests to a different destination. **Lpmove** will refuse to move requests while the LP scheduler is running.

#### EXAMPLES

1. Move all requests for printer abc to printer xyz:

```
$ /usr/lib/lpmove abc xyz
```

   All of the moved requests are renamed from abc-nnn to xyz-nnn. As a side effect, destination abc is no longer accepting further requests.

2. Move requests zoo-543 and abc-1200 to printer xyz:

```
$ /usr/lib/lpmove zoo-543 abc-1200 xyz
```

   The two requests are now renamed xyz-543 and xyz-1200.

### STOPPING AND STARTING THE SCHEDULER—LPSHUT AND LPSCHED

**Lpsched** is the program that routes the output requests that were made with **lp** through the appropriate printer interface programs to be printed on line printers. Each time the scheduler routes a request to an interface program, it records an entry in the log file, */usr/spool/lp/log*. This entry contains the logname of the user that made the request, the request id, the name of the printer that the request is being printed on, and the date and time that printing first started. In the case that a request has been restarted, more than one entry in the log file may refer to the request. The scheduler also records error messages in the log file. When **lpsched** is started, it renames */usr/spool/lp/log* to */usr/spool/lp/oldlog* and starts a new log file.

No printing will be performed by the LP system unless **lpsched** is running. Use the command

```
lpstat −r
```

to find the status of the LP scheduler.

**Lpsched** is normally started by the **/etc/rc** program as described above and continues to run until the UNIX system is shut down. The scheduler operates in the */usr/spool/lp* directory. When it starts running, it will exit immediately if a file called *SCHEDLOCK* exists. Otherwise, it creates this file in order to prevent more than one scheduler from running at the same time.

Occasionally, it is necessary to shut down the scheduler in order to reconfigure LP or to rebuild the LP software. The command

```
/usr/lib/lpshut
```

causes **lpsched** to stop running and terminates all printing activity. All requests that were in the middle of printing will be reprinted in their entirety when the scheduler is restarted.

To restart the LP scheduler, use the command

```
/usr/lib/lpsched
```

Shortly after this command is entered, **lpstat** should report that the scheduler is running. If not, it is possible that a previous invocation of **lpsched** exited without removing *SCHEDLOCK*, so try the following:

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
```

The scheduler should be running now.

### PRINTER INTERFACE PROGRAMS

Every LP printer must have an interface program which does the actual printing on the device that is currently associated with the printer. Interface programs may be shell procedures, C programs, or any other executable programs. The LP model interfaces are all written as shell procedures and can be found in the */usr/spool/lp/model* directory. At the time **lpsched** routes an output request to a printer P, the interface program for P is invoked in the directory */usr/spool/lp* as follows:

```
interface/P id user title copies options file ...
where
id is the request id returned by lp
user is logname of user who made the request
title is optional title specified by the user
copies is number of copies requested by user
options is a blank-separated list of class or
printer-dependent options specified by user
file is the full pathname of a file to be printed
```

## EXAMPLES

The following examples are requests made by user "smith" with a system default destination of printer "xyz". Each example lists an **lp** command line followed by the corresponding command line generated for printer xyz's interface program:

1. lp /etc/passwd /etc/group
   interface/xyz xyz-52 smith " " 1 " " /etc/passwd /etc/group

2. pr /etc/passwd | lp -t " users " -n5
   interface/xyz xyz-53 smith users 5 " "
   /usr/spool/lp/request/xyz/d0-53

3. lp /etc/passwd -oa -ob
   interface/xyz xyz-54 smith " " 1 " a b " /etc/passwd

When the interface program is invoked, its standard input comes from */dev/null* and both the standard output and standard error output are directed to the printer's device. Devices are opened for reading as well as

writing when file modes permit. In the case where a device is a regular file, all output is appended to the end of the file.

Given the command line arguments and the output directed to a device, interface programs may format their output in any way they choose. Interface programs must ensure that the proper stty modes (terminal characteristics such as baud rate, output options, etc.) are in effect on the output device. This may be done as follows in a shell interface only if the device is opened for reading:

```
stty mode ... <&1
```

That is, take the standard input for the **stty** command from the device.

When printing has completed, it is the responsibility of the interface program to exit with a code indicative of the success of the print job. Exit codes are interpreted by **lpsched** as follows:

| CODE | MEANING TO LPSCHED |
|---|---|
| zero | The print job has completed successfully. |
| 1 to 127 | A problem was encountered in printing this particular request (e.g., too many nonprintable characters). This problem will not affect future print jobs. **Lpsched** notifies users by mail that there was an error in printing the request. |
| greater than 127 | These codes are reserved for internal use by **lpsched**. Interface programs must not exit with codes in this range. |

When problems that are likely to affect future print jobs occur (e.g., a device filter program is missing), the interface programs would be wise to disable printers so that print requests are not lost. When a busy printer is disabled, the interface program will be terminated with signal 15.

## SETTING UP HARDWIRED DEVICES AND LOGIN TERMINALS AS LP PRINTERS

### A. Hardwired Devices

As an example of how to set up a hardwired device for use as an LP printer, let us consider using tty line 15 as printer xyz. As superuser, perform the following:

1. Avoid unwanted output from non-LP processes and ensure that LP can write to the device:

   ```
   $ chown lp /dev/tty15
   $ chmod 600 /dev/tty15
   ```

2. Change */etc/inittab* so that tty15 is not a login terminal. In other words, ensure that */etc/getty* is not trying to log users in at this terminal. Change the entries for line 15 to:

   ```
   1:15:o:
   2:15:o:
   ```

   Enter the command:

   ```
   $ init 2
   ```

   If there is currently an invocation of */etc/getty* running on tty15, kill it. Now, and when the UNIX system is rebooted, tty15 will be initialized with default stty modes. Thus, it is up to LP interface programs to establish the proper baud rate and other stty modes for correct printing to occur.

3. Introduce printer xyz to LP using the model prx interface program:

   $ /usr/lib/lpadmin −pxyz −v/dev/tty15 −mprx

4. When xyz is created, it will initially be disabled and **lp** will be rejecting requests routed to it. If it is desired, allow **lp** to accept requests for xyz:

   /usr/lib/accept xyz

   This will allow requests to build up for xyz, and to be printed when it is enabled at a later time.

5. When it is desired for printing to occur, be sure that the printer is ready to receive output. For several printers, this means that the top of form has been adjusted and that the printer is on-line. Enable printing to occur on xyz:

   enable xyz

   When requests have been routed to xyz, they will begin printing.

### B. Login Terminals

Login terminals may also be used as LP printers. To do this for a Diablo 1640 terminal called abc, perform the following:

1. Introduce printer abc to LP using the model 1640 interface program:

   $ /usr/lib/lpadmin −pabc −v/dev/null −m1640 −l

   Note that */dev/null* is used as abc's device because we will specify the actual device each time that abc is enabled. This device may be different from day to day. When abc is created, it will initially be disabled; and **lp** will be rejecting requests routed to it. If it is desired, allow **lp** to accept requests for abc:

   /usr/lib/accept abc

   This will allow requests to build up for abc and to be printed when it is enabled at a later time. It is not advisable to enable abc for printing, however, until the following steps have been taken.

2. Log terminal in if this has not already been done.

3. Assuming the **tty**(1) command reports that this terminal is */dev/tty02*, associate this device with printer abc:

   $ /usr/lib/lpadmin −pabc −v/dev/tty02

   Note that **lpadmin** may be used only by an LPA. If it is desired for other users to routinely perform this step, then an LPA may establish a program owned by **lp** or by **root** with set-user-id permission that performs this function.

4. When it is desired for printing to occur, be sure that the printer is ready to receive output. For several printers, this means that the top of form has been adjusted. Enable printing to occur on abc:

   enable abc

   When requests have been routed to abc, they will begin printing.

5. When all printing has stopped on abc or when you want it back as a regular login terminal, you may prevent it from printing more output:

```
$ disable abc
printer "abc" now disabled
```

If abc is enabled when the UNIX system is rebooted or when **lpsched** is restarted, it will be disabled automatically.

## SUMMARY

The administrative functions of the LP administrator have been described in detail. These functions include configuring and reconfiguring LP; maintaining printer interface programs; accepting, rejecting, and moving print requests; stopping and starting the LP scheduler; and enabling and disabling printers. LP offers administrators the following advantages over other centrally supported printer packages:

- Printers may be grouped into classes.
- LP may be configured to meet the needs of each site.
- Administrators may supply interface programs to format output in any way desirable.
- LP functions are performed by simple commands and not by hand.

*NOTES*

# UUCP

# Administrator's Manual

Trademarks:

| | |
|---|---|
| MUNIX, CADMUS | for PCS |
| DEC, PDP | for DEC |
| UNIX | for Bell Laboratories |

CONTENTS

# Uucp Administrator's Manual

## 1. Introduction

*Uucp* has been the mainstay of UNIX[1] system to UNIX system communication for several years [1, 2]. This document illustrates how a network is set up, the format of control files and administrative procedures. Administrators should be familiar with the *Uucp Users Tutorial* and the manual pages for each of the *uucp* related commands before reading this document.

## 2. Planning

In setting up a network of UNIX systems there are several considerations that should be taken into account *before* configuring each system on the network. The following sections attempt to outline the most important considerations.

### 2.1 Extent of the Network

Some basic decisions about *access* to processors in the network must be made before attempting to set up the configuration files. If an administrator has control over only one processor and an existing network is being joined, then the administrator must decide what level of access should be granted to other systems. The other members of the network must make a similar decision for the new system. The UNIX system *password* mechanism is used to grant access to other systems. The file */usr/lib/uucp/USERFILE)* restricts access by other systems to parts of the filesystem tree and the file */usr/lib/uucp/L.sys* on the local processor determines how many other systems on the network can be reached.

When setting up more than one processor is involved, the administrator has control of a larger fraction of the network and can make more decisions about the setup of the network. For example, the network can be set up as a *private* network where only those machines under the direct control of the-administrator can access each other. Granting *no* access to machines outside the network can be done if security is paramount, however, this is usually impractical. Very limited access can be granted to outside machines by each of the systems on the *private* network. Alternatively, access to/from the outside world can be confined to only one processor. This is frequently done to minimize the effort in keeping access information (passwords, phone numbers, login sequences, etc.) updated and to minimize the number of security holes for the private network.

### 2.2 Hardware and Line Speeds

There are only two supported means of interconnection by *uucp(1)*,

1. Direct connection using a null modem.
2. Connection over the DDD network.

Direct connection over private lines using X.25 is not fully supported in UNIX System V, although with the addition of one program *x25.login* it can be made operational. In choosing hardware, the equipment used by other processors on the network must be considered. For example, if some systems on the network have only 103 type (300 baud) datasets, then communication with them is not possible unless the local system has a 300 baud dataset connected to a calling unit. (Most datasets available on systems are 1200 baud.) If hardwired connections are to be used between systems, then the *distance* between systems must be considered since a null modem cannot be used when the systems are separated by more than several hundred feet (the limit for communication at

---

* UNIX is a Trademark of Bell Telephone Laboratories, Incorporated.

9600 baud is about 800 to 1000 feet although the RS232 specification allows for less than fifty feet). Limited distance modems must be used beyond these distances or if noise on the lines becomes a problem.

### 2.3 Maintenance and Administration

There is a minimum amount of maintenance that must be provided on each system to keep the access files updated, to insure that the network is running properly and to track down line problems. When more than one system is involved, the job becomes more difficult because there are more files to update and because users are much less patient when failures occur between machines that are under local control.

## 3. A Quick Tour of the Uucp Software

Figure 1 is an illustration of the daemons used by the *uucp* network to communicate with another system. The *uucp(1)* or *uux(1)* command queues users requests and spawns the *uucico* daemon to call another system. Figure 2 illustrates the structure of *uucico* and the tasks that it performs in communicating with another system. It initiates the call to another system and performs the file transfer. On the receiving side, *uucico* is invoked to receive the transfer. Remote execution jobs are actually done by transferring a command file to the remote system and invoking a daemon *(uuxqt)* to execute that command file and return the results.

## 4. Installation

The *uucp(1)* package is delivered as part of the standard UNIX system distribution. It resides in its own subdirectory (called *uucp)* in the commands area and has its own make file *(uucp.mk)*. The *uucp* package is installed as part of the normal distribution, however, if it must be reinstalled for any reason, then the sequence

*make uucp.mk install*

should be executed.

### 4.1 Object Modules

The following object modules are installed as part of the *uucp* make procedure,

1. *Uucp* - The file transfer command.
2. *Uux* - The remote execution command.
3. *Uucico* - The *uucp* network daemon.
4. *Uustat* - Network status command.
5. *Uuclean* - Cleanup command.
6. *Uusub* - The command for monitoring and creating a subnetwork.
7. *Uuxqt* - The remote execution daemon.
8. *Uudemon.day* - A shell procedure that is invoked each day to maintain the network. Shell scripts for execution each week *(uudemon.wk)* and each hour *(uudemon.hr)* are also distributed.

### 4.2 Password FIle

To allow remote systems to call the local system, password entries must be made for any *uucp* logins. For example,

nuucp:zaaAA:6:1:45422-UUCP.Admin:/usr/spool/uucppublic:/usr/lib/uucp/uucico

Note that the *uucico* daemon is used for the *shell* and the spool directory is used as the working directory.

**4.3 Lines File**

The file */usr/lib/uucp/L-devices* contains the list of all lines that are directly connected to other systems or are available for calling other systems. The file contains the attributes of the lines and whether the line is a permanent connection or can call via a dialer. The format of the file is

*type line call-device speed protocol*

where each field is

*type* Two keywords are used to describe whether a line is directly connected to another system (DIR) or uses an automatic calling unit (ACU). An X.25 permanent virtual circuit would use the DIR keyword.

*line* This is the device name for the line (e.g., *ttyab* for a direct line, *cul0* for a line connected to an ACU).

*call-device* If the ACU keyword is specified, this field contains the device name of the automatic calling unit. Otherwise, the field is ignored, however, a placeholder must be used in this field so that the *protocol* field can be interpreted.

*speed* The line speed that the connection is to run at. (The speed field is currently ignored if an X.25 link is used.)

*protocol* This is an optional field that needs only be filled in if the connection is for a protocol other than the default terminal protocol. The X.25 protocol is the only other protocol supported and the single character *x* is used to select this protocol.

The following entries illustrate various types of connections,

```
DIR ttyab 0 9600
ACU cul0 cua0 1200
DIR x25.s0 0 300 x
```

The first entry is for a hardwired line running at 9600 baud between two systems. Note that the *acu-device* field is zero. The second entry is for an line with a 1200 baud automatic calling unit. The last entry is for an X.25 synchronous direct connection between systems. Note that the *protocol* field is filled in and that the *acu-device* and *line speed* fields are meaningless.

*4.3.1 Naming Conventions* It is often useful when naming lines that are directly connected between systems or which are dedicated to calling other systems to choose a naming scheme that conveys the use of the line. In the earlier examples, the name *ttyab* is used for the line that directly connects two systems named *a* and *b*. Similarly, lines associated with calling units are best given names that relate them to the their calling unit (note the names *cul0* and *cua0* to specify the line and calling unit respectively).

**4.4 System File**

Each entry in this file represents a system that can be called by the local *uucp* programs. More than one line may be present for a particular system. In this case, the additional lines represent alternative communication paths that will be tried in sequential order. The fields are described below.

*system name* The name of the remote system.

*time* This is a string that indicates the days-of-week and times-of-day when the system should be called (e.g., MoTuTh0800−1730).

The day portion may be a list containing some of *Su Mo Tu We Th Fr Sa* or it may be *Wk* for any week-day or *Any* for any day. The time should be a range of times (e.g., 0800−1230). If no time portion is specified, any time of day is assumed to be allowed for the call. Note that a time range that spans 0000 is permitted, for example, 0800-0600 means all times are allowed other than times between 6 and 8 am. An optional subfield is available to specify the minimum time (minutes) before a retry following a failed attempt. The subfield separator is a "," (e.g., Any,9 means call any time but wait at least 9 minutes before retrying the call after a failure has occurred).

*device* This is either *ACU* or the hard-wired device name to be used for the call. For the hard-wired case, the last part of the special file name is used (e.g., tty0).

*class* This is usually the line speed for the call (e.g., 300).

*phone* The phone number is made up of an optional alphabetic abbreviation (dialing prefix) and a numeric part. The abbreviation should be one that appears in the *L-dialcodes* file (e.g., mh5900, boston995−9980). For the hard-wired devices, this field contains the same string as used for the *device* field (e.g., tty0, etc.).

*login* The login information is given as a series of fields and subfields in the format

[ expect send ] ...

where *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The expect field may be made up of subfields of the form

expect[−send−expect] ...

where the *send* is sent if the prior *expect* is *not* successfully read and the *expect* following the *send* is the next expected string. (e.g., login--login will expect *login*; if it gets it, the program will go on to the next field; if it does not get *login*, it will send *null* followed by a new line, then expect *login* again.)

There are two special names available to be sent during the login sequence. The string *EOT* will send an EOT character and the string *BREAK* will try to send a *BREAK* character. (The *BREAK* character is simulated using line speed changes and null characters and may not work on all devices and/or systems.) A number from 1 to 9 may follow the *BREAK* for example, *BREAK1*, will will send 1 null character instead of the default of 3. Note that *BREAK1* usually works best for 300/1200 baud lines.

A typical entry in the L.sys file would be

sys Any ACU 300 mh7654 login uucp ssword: word

The expect algorithm matches all or part of the input string as illustrated in the password field above.

**4.5 Dialing Prefixes**

This file contains the dial-code abbreviations used in the *L.sys* file (e.g., py, mh, boston). The entry format is

abb dial-seq

where

| | |
|---|---|
| abb | is the abbreviation, |
| dial-seq | is the dial sequence to call that location. |

The line

py 165–

would be set up so that entry py7777 would send 165–7777 to the dial-unit.

**4.6 USERFILE**

This file contains user accessibility information. It specifies four types of constraint,

[1] which files can be accessed by a normal user of the local machine,

[2] which files can be accessed from a remote computer,

[3] which login name is used by a particular remote computer,

[4] whether a remote computer should be called back in order to confirm its identity.

Each line in the file has the format

login,sys [ c ] path-name [ path-name ] ...

where

| | |
|---|---|
| login | is the login name for a user or the remote computer, |
| sys | is the system name for a remote computer, |
| c | is the optional *call-back required* flag, |
| path-name | is a path-name prefix that is acceptable for *sys*. |

The constraints are implemented as follows.

[1] When the program is obeying a command stored on the local machine, the path-names allowed are those given on the first line in the *USERFILE* that has the login name of the user who entered the command. If no such line is found, the first line with a *null* login name is used.

[2] When the program is responding to a command from a remote machine, the path-names allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a *null* system name is used.

[3] When a remote computer logs in, the login name that it uses *must* appear in the *USERFILE*. There may be several lines with the same login name but one of

them must either have the name of the remote system or must contain a *null* system name.

[4] If the line matched in ([3]) contains a "c", the remote machine is called back before any transactions take place.

The line

```
u,m  /usr/xyz
```

allows machine *m* to login with name *u* and request the transfer of files whose names start with "/usr/xyz". The line

```
you,  /usr/you
```

allows the ordinary user *you* to issue commands for files whose name starts with "/usr/you". (Note that this type restriction is seldom used.) The lines

```
u,m /usr/xyz  /usr/spool
u,  /usr/spool
```

allows *any* remote machine to login with name *u*. If its system name is not *m*, it can only ask to transfer files whose names start with "/usr/spool". If it is system *m*, it can send files from paths "/usr/xyz" as well as "/usr/spool". The lines

```
root,  /
,   /usr
```

allow any user to transfer files beginning with "/usr" but the user with login *root* can transfer any file. (Note that any file that is to be transferred must be readable by anybody.)

**4.7 Forwarding File**

There are two files that allow restrictions to be placed on the forwarding mechanism. The format of the entries in each file is the same,

```
system
```

or

```
system!user!user2,...
```

The file *ORIGFILE (/usr/lib/uucp/ORIGFILE)* restricts the access of systems that are attempting to forward *through the local system.* The file contains the list of systems (and users) for whom the local system is willing to forward. Each entry refers to the system that was the *source* of the original job and not the name of the last system to forward the file. The second file *FWDFILE (/usr/lib/uucp/FWDFILE)* is a list of valid systems that a job can be *forwarded to* (it is not necessarily the name of the destination of a job, but merely the next valid node). This file will be a subset of the L.sys file and can be used to prevent forwarding to systems that are very expensive to reach, but to which access by local users is allowed (for example, links to overseas universities). If neither of these files exist, *uucp* will be perfectly happy to forward for any system. As an example, if the entry for system *australia* were in the ORIGFILE but not in the FWDFILE on system xnode, it would mean that system *australia* would be capable of forwarding jobs into the network via system xnode however, no systems in the network could forward a job to *australia* via system xnode.

## 5. Administration

The role of the *uucp* administrator depends heavily on the amount of traffic that enters or leaves a system and the quality of the connections that can be made to and from that system. For the average system, only a modest amount of traffic (100-200 files per day) pass through the system and little if any intervention with the *uucp* automatic cleanup functions is necessary. Systems that pass large numbers of files (200-10000) may require more attention when problems occur. The following sections describe the routine administrative tasks that must be performed by the administrator or are automatically performed by the *uucp* package. The section on problems describes what are the most frequent problems and how to effectively deal with them.

### 5.1 Cleanup

The biggest problem in a dialup network like *uucp* is dealing with the backlog of jobs that cannot be transmitted to other systems. The following cleanup activities should be routinely performed by shell scripts started from *cron(1)*.

*5.1.1 Cleanup of Undeliverable Jobs* The *uudemon.day* procedure usually contains an invokation of the *uuclean* commany to purge any jobs that are older than some fixed time (usually 72 hours). A similar procedure is usually used to purge any *lock* or *status* files. An example invokation of *uuclean(1m)* to remove both job files and old status files every 48 hours is

*/usr/lib/uuclean -pST -pC -n48*

*5.1.2 Cleanup of the Public Area* In order to keep the local filesystem from overflowing when files are sent to the public area, the *uudemon.day* procedure is usually set up with a *find* command to remove any files that are older than seven days. This interval may need to be shortened if there is not sufficient space to devote to the public area.

*5.1.3 Compaction of Log Files* The files *SYSLOG* and *LOGFILE* that contain logging information are compacted daily (using the *pack* command) and should be kept for one week before being overwritten.

### 5.2 Polling Other Systems

Systems that are passive members of the network must be polled by other systems in order for their files to be sent. This can be arranged by using the *uusub(1)* command as follows,

*uusub -ccnode*

which will call *cnode* when it is invoked.

### 5.3 Problems

The following sections list the most frequent problems that appear on systems that make heavy use of *uucp(1)*.

*5.3.1 Out of Space* The filesystem used to spool incoming or outgoing jobs can run out of space and prevent jobs from being spawned or much worse received from remote systems. The inability to receive jobs is the worse of the two conditions since when filespace does become available, the system will be *flooded* with the backlog of traffic.

*5.3.2 Bad Acu's and Modems* The automatic calling units and incoming modems occasionally cause problems that make it difficult to contact other systems or to receive files. These problems are usually readily identifiable since *LOGFILE* entries will usually point to the bad line. If a bad line is suspected, it it useful to use the *cu(1)* command to try calling another system using the suspected line.

*5.3.3 Administrative problems* Some *uucp* networks have so many members that it is difficult to keep track of changing passwords, changing phone numbers or changing logins on remote systems. This can be a very costly problem since acu's will be tied up calling a system that cannot be reached.

**6. Debugging**

In order to verify that a system on the network can be contacted, the *uucico* daemon can be invoked from a user's terminal directly. For example, to verify that *cnode* can be contacted, a job would be queued for that system as follows,

*uucp -r file cnode!~hom*

The *-r* option forces the job to be queued but does not invoke the daemon to process the job. The *uucico* command can then be invoked directly,

*/usr/lib/uucp/uucico -r1 -x4 -scnode*

The *-r1* option is necessary to indicate that the daemon is to start up in *master* mode (that is, it is the calling system). The *-x4* specifies the level of debugging that is to be printed. Higher levels of debugging can be printed (greater than 4) but requires familiarity with the internals of *uucico*. If several jobs are queued for the remote system, it is not possible to force *uucico* to send one particular job first. The contents of LOGFILE should also be monitored for any error indications that it posts. Frequently, problems can be isolated by examining the entries in LOGFILE associated with a particular system. The file *ERRLOG* also contains error indications.

**7. Conclusion**

This manual has emphasized the format of control files and some of the issues in setting up a *uucp* network.

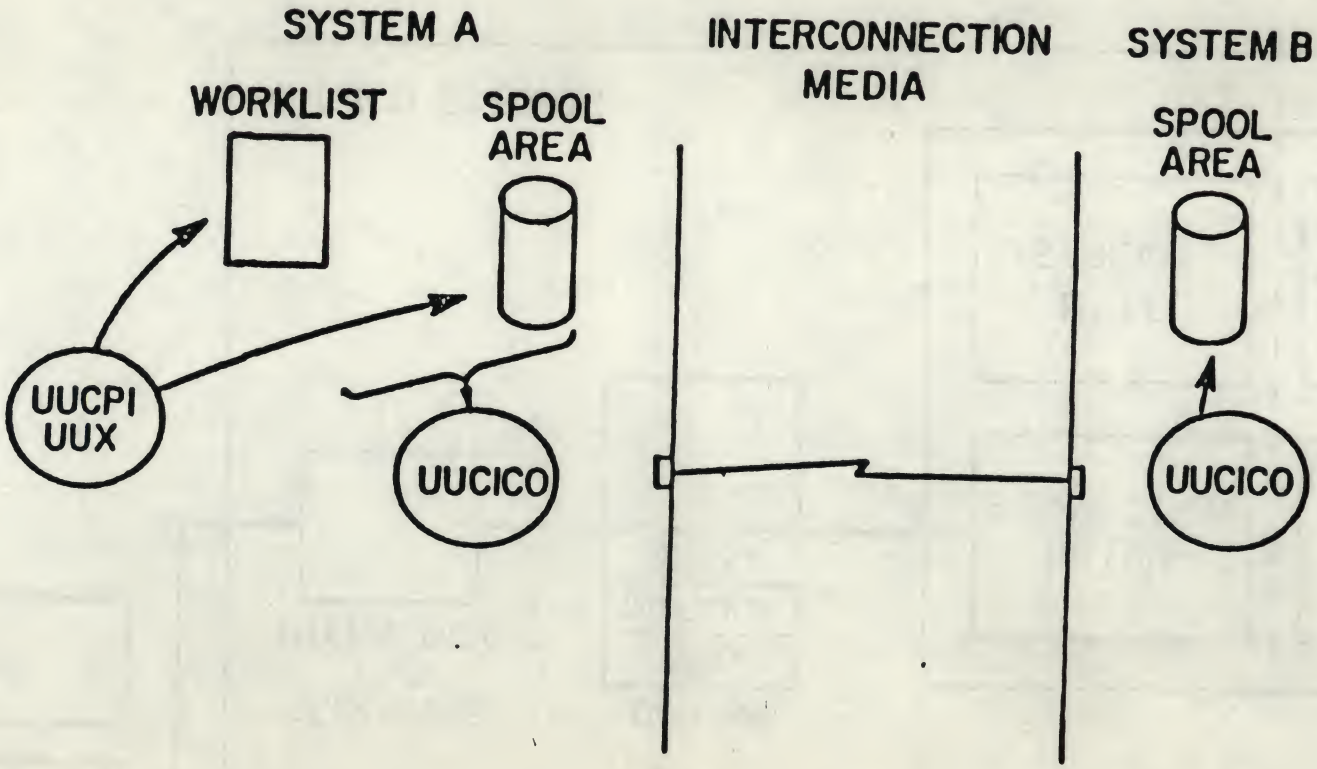


Figure 1 Uucp network daemons.

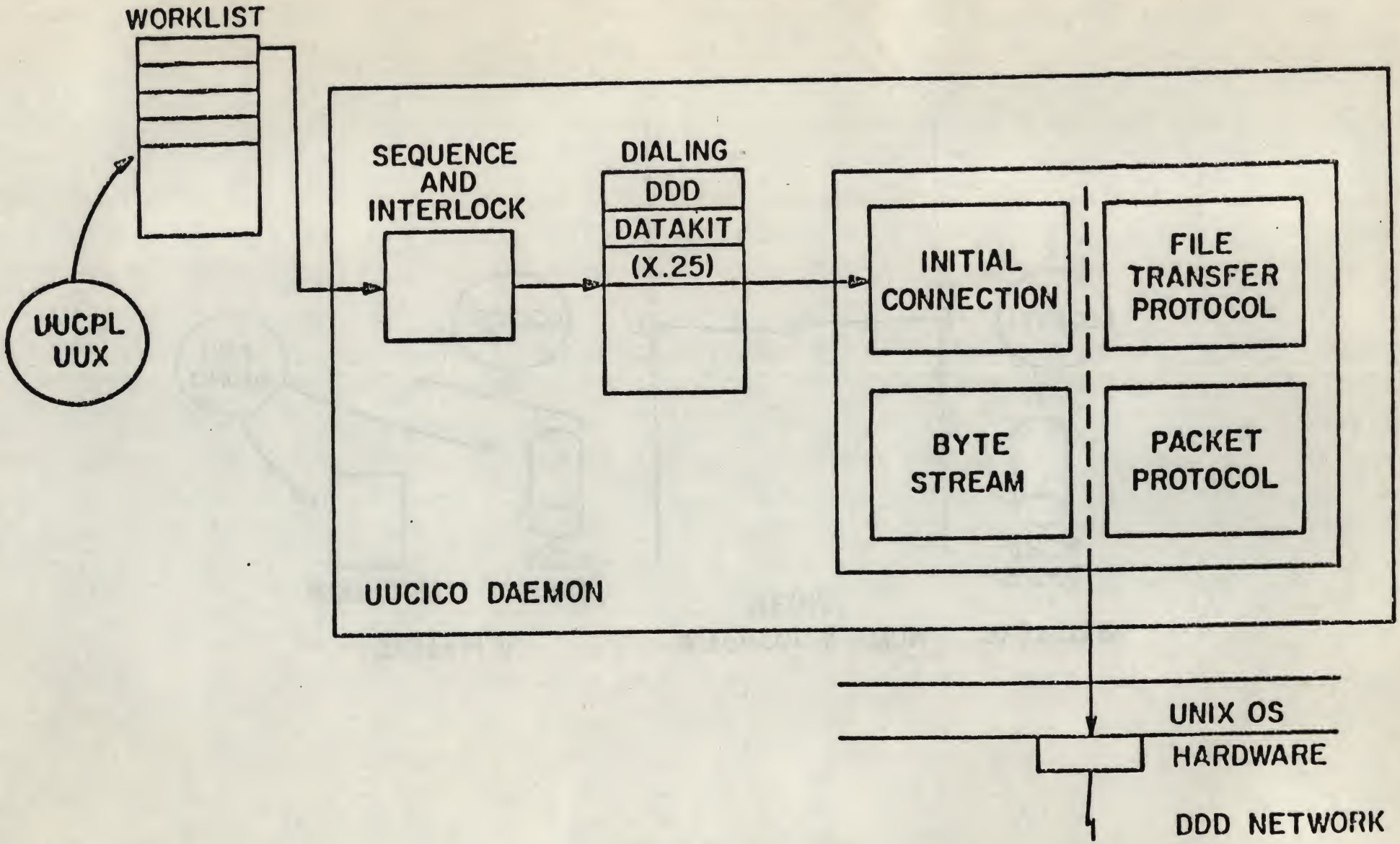


Figure 2 Uucico daemon functional blocks.

# UNIX System Accounting

This document is part of the ADMINISTRATOR'S GUIDE. Therefore the pagenumbers don't begin with 1.

## 7. UNIX SYSTEM ACCOUNTING

The UNIX System Accounting provides methods to collect per-process resource utilization data, record connect sessions, monitor disk utilization, and charge fees to specific logins. A set of C language programs and shell procedures is provided to reduce this accounting data into summary files and reports. This section describes the structure, implementation, and management of this accounting system, as well as a discussion of the reports generated and the meaning of the columnar data.

### GENERAL

The following list is a synopsis of the actions of the accounting system:

- At process termination, the UNIX system kernel writes one record per process in */usr/adm/pacct* in the form of *acct.h*. (See Attachment 7.1 for a description of data files.)
- The **login** and **init** programs record connect sessions by writing records into */etc/wtmp*. Date changes, reboots, and shutdowns are also recorded in this file.
- The disk utilization program **acctdusg** breaks down disk usage by login.
- Fees for file restores, etc., can be charged to specific logins with the **chargefee** shell procedure.
- Each day the **runacct** shell procedure is executed via **cron** to reduce accounting data and produce summary files and reports. (See Attachment 7.2 for a sample report output.)
- The **monacct** procedure can be executed on a monthly or fiscal period basis. It saves and restarts summary files, generates a report, and cleans up the *sum* directory. These saved summary files could be used to charge users for UNIX system usage.

### FILES AND DIRECTORIES

The */usr/lib/acct* directory contains all of the C language programs and shell procedures necessary to run the accounting system. The *adm* login (currently user ID of four) is used by the accounting system and has the directory structure shown in Fig. 7.1.

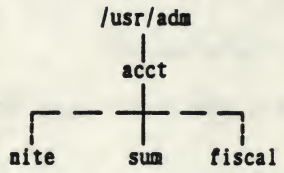


**Fig. 7.1—Directory Structure of the "adm" Login**

The */usr/adm* directory contains the active data collection files. (For a complete explanation of the files used by the accounting system, see Attachment 7.3.) The *nite* directory contains files that are reused daily by the **runacct** procedure. The *sum* directory contains the cumulative summary files updated by **runacct**. The *fiscal* directory contains periodic summary files created by **monacct**.

## DAILY OPERATION

When the UNIX system is switched into multiuser mode, */usr/lib/acct/startup* is executed which does the following:

1. The **acctwtmp** program adds a "boot" record to */usr/adm/wtmp*. This record is signified by using the system name as the login name in the *wtmp* record.

2. Process accounting is started via **turnacct**. **Turnacct on** executes the **accton** program with the argument */usr/adm/pacct*.

3. The **remove** shell procedure is executed to clean up the saved *pacct* and *wtmp* files left in the *sum* directory by **runacct**.

The **ckpacct** procedure is run via **cron** every hour of the day to check the size of */usr/adm/pacct*. If the file grows past 1000 blocks (default), **turnacct switch** is executed. While **ckpacct** is not absolutely necessary, the advantage of having several smaller *pacct* files becomes apparent when trying to restart **runacct** after a failure processing these records.

The **chargefee** program can be used to bill users for file restores, etc. It adds records to */usr/adm/fee* which are picked up and processed by the next execution of **runacct** and merged into the total accounting records.

**Runacct** is executed via **cron** each night. It processes the active accounting files, */usr/adm/pacct*, */usr/adm/wtmp*, */usr/adm/acct/nite/disktacct*, and */usr/adm/fee*. It produces command summaries and usage summaries by login.

When the system is shut down using **shutdown**, the **shutacct** shell procedure is executed. It writes a shutdown reason record into */usr/adm/wtmp* and turns process accounting off.

After the first reboot each morning, the computer operator should execute */usr/lib/acct/prdaily* to print the previous day's accounting report.

## SETTING UP THE ACCOUNTING SYSTEM

In order to automate the operation of this accounting system, several things need to be done:

1. If not already present, add this line to the */etc/rc* file in the state 2 section:

```
/bin/su -adm -c /usr/lib/acct/startup
```

2. If not already present, add this line to */etc/shutdown* to turn off the accounting before the system is brought down:

```
/usr/lib/acct/shutacct
```

3. For most installations, the following three entries should be made in */usr/lib/crontab* so that **cron** will automatically run the daily accounting.

```
" 0 4 * * 1-6 /bin/su -adm -c " /usr/lib/acct/runacct
        2>    /usr/adm/acct/nite/fd2log "
  0 2 * * 4 /usr/lib/acct/dodisk
  5 * * * * /bin/su -adm -c " /usr/lib/acct/ckpacct "
```

Note that **dodisk** is invoked with superuser privileges of *root* so that directory searching is not road blocked.

4. To facilitate monthly merging of accounting data, the following entry in *crontab* will allow **monacct** to clean up all daily reports and daily total accounting files and deposit one monthly total report and one monthly total accounting file in the *fiscal* directory.

```
15 5 1 * * /bin/su -adm -c /usr/lib/acct/monacct
```

The above entry takes advantage of the default action of **monacct** that uses the current month's date as the suffix for the file names. Notice that the entry is executed at such a time as to allow **runacct** sufficient time to complete. This will, on the first day of each month, create monthly accounting files with the entire month's data.

5. The *PATH* shell variable should be set in */usr/adm/.profile* to:

```
PATH=/usr/lib/acct:/bin:/usr/bin
```

## RUNACCT

**Runacct** is the main daily accounting shell procedure. It is normally initiated via **cron** during nonprime time hours. **Runacct** processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by **prdaily** or for billing purposes. The following files produced by **runacct** are of particular interest:

| | |
|---|---|
| nite/lineuse | Produced by **acctcon**, which reads the *wtmp* file, and produces usage statistics for each terminal line on the system. This report is especially useful for detecting bad lines. If the ratio between the number of logoffs to logins exceeds about 3/1, there is a good possibility that the line is failing. |
| nite/dayacct | This file is the total accounting file for the previous day in **tacct.h** format. |
| sum/tacct | This file is the accumulation of each day's *nite/daytacct* which can be used for billing purposes. It is restarted each month or fiscal period by the **monacct** procedure. |
| sum/daycms | Produced by the **acctcms** program, it contains the daily command summary. The ASCII version of this file is *nite/daycms*. |
| sum/cms | The accumulation of each day's command summaries. It is restarted by the execution of **monacct**. The ASCII version is *nite/cms*. |
| sum/loginlog | Produced by the **lastlogin** shell procedure, it maintains a record of the last time each login was used. |
| sum/rprt.MMDD | Each execution of **runacct** saves a copy of the output of **prdaily**. |

**Runacct** takes care not to damage files in the event of errors. A series of protection mechanisms are used that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that **runacct** can be restarted with minimal intervention. It records its progress by writing descriptive messages into the file *active*. (Files used by **runacct** are assumed to be in the *nite* directory unless otherwise noted.) All diagnostics output during the execution of **runacct** is written into *fd2log*. To prevent multiple invocations, in the event of two **crons** or other problems, **runacct** will complain if the files *lock* and *lock1* exist when invoked. The *lastdate* file contains the month and day **runacct** was last invoked and is used to prevent more than one execution per day. If **runacct** detects an error, a message is written to */dev/console*, mail is sent to *root* and *adm*, the locks are removed, diagnostic files are saved, and execution is terminated.

In order to allow **runacct** to be restartable, processing is broken down into separate reentrant states. This is accomplished by using a **case** statement inside an endless **while** loop. Each state is one case of the **case**

statement. A file is used to remember the last state completed. When each state completes, *statefile* is updated to reflect the next state. In the next loop through the **while**, *statefile* is read and the **case** falls through to the next state. When **runacct** reaches the **CLEANUP** state, it removes the locks and terminates. States are executed as follows:

| | |
|---|---|
| SETUP | The command **turnacct switch** is executed. The process accounting files, */usr/adm/pacct?*, are moved to */usr/adm/Spacct?.MMDD*. The */usr/adm/wtmp* file is moved to */usr/adm/acct/nite/wtmp.MMDD* with the current time added on the end. |
| WTMPFIX | The *wtmp* file in the *nite* directory is checked for correctness by the **wtmpfix** program. Some date changes will cause **acctcon1** to fail, so **wtmpfix** attempts to adjust the time stamps in the *wtmp* file if a date change record appears. |
| CONNECT1 | Connect session records are written to *ctmp* in the form of **ctmp.h**. The *lineuse* file is created, and the *reboots* file is created showing all of the boot records found in the *wtmp* file. |
| CONNECT2 | *Ctmp* is converted to *ctacct.MMDD* which are connect accounting records. (Accounting records are in **tacct.h** format.) |
| PROCESS | The **acctprc1** and **acctprc2** programs are used to convert the process accounting files, */usr/adm/Spacct?.MMDD*, into total accounting records in *ptacct?.MMDD*. The *Spacct* and *ptacct* files are correlated by number so that if **runacct** fails, the unnecessary reprocessing of *Spacct* files will not occur. One precaution should be noted; when restarting **runacct** in this state, remove the last *ptacct* file because it will not be complete. |
| MERGE | Merge the process accounting records with the connect accounting records to form *daytacct*. |
| FEES | Merge in any ASCII *tacct* records from the file *fee* into *daytacct*. |
| DISK | On the day after the **sdisk** procedure runs, merge *disktacct* with *daytacct*. |
| MERGETACCT | Merge *daytacct* with *sum/tacct*, the cumulative total accounting file. Each day, *daytacct* is saved in *sum/tacctMMDD*, so that *sum/tacct* can be recreated in the event it becomes corrupted or lost. |
| CMS | Merge in today's command summary with the cumulative command summary file *sum/cms*. Produce ASCII and internal format command summary files. |
| USEREXIT | Any installation dependent (local) accounting programs can be included here. |
| CLEANUP | Clean up temporary files, run **prdaily** and save its output in *sum/rprtMMDD*, remove the locks, then exit. |

### RECOVERING FROM FAILURE

The **runacct** procedure can fail for a variety of reasons; usually due to a system crash, */usr* running out of space, or a corrupted *wtmp* file. If the *activeMMDD* file exists, check it first for error messages. If the *active* file and lock files exist, check *fd2log* for any mysterious messages. The following are error messages produced by **runacct**, and the recommended recovery actions:

ERROR: locks found, run aborted

The files *lock* and *lock1* were found. These files must be removed before **runacct** can restart.

ERROR: acctg already run for *date*: check /usr/adm/acct/nite/lastdate

The date in *lastdate* and today's date are the same. Remove *lastdate*.

ERROR: turnacct switch returned rc=?

Check the integrity of **turnacct** and **accton**. The **accton** program must be owned by *root* and have the *setuid* bit set.

ERROR: Spacct?.*MMDD* already exists

File setups probably already run. Check status of files, then run setups manually.

ERROR: /usr/adm/acct/nite/wtmp.*MMDD* already exists, run setup manually

Self-explanatory.

ERROR: wtmpfix errors see /usr/adm/acct/nite/wtmperror

**Wtmpfix** detected a corrupted *wtmp* file. Use **fwtmp** to correct the corrupted file.

ERROR: connect acctg failed: check /usr/adm/acct/nite/log

The **acctcon1** program encountered a bad *wtmp* file. Use **fwtmp** to correct the bad file.

ERROR: Invalid state, check /usr/adm/acct/nite/active

The file *statefile* is probably corrupted. Check *statefile* and read *active* before restarting.

### RESTARTING RUNACCT

**Runacct** called without arguments assumes that this is the first invocation of the day. The argument *MMDD* is necessary if **runacct** is being restarted and specifies the month and day for which **runacct** will rerun the accounting. The entry point for processing is based on the contents of *statefile*. To override *statefile*, include the desired state on the command line. For example:

To start **runacct**:

```
nohup runacct  2>  /usr/adm/acct/nite/fd2log&
```

To restart **runacct**:

```
nohup runacct 0601  2>  /usr/adm/acct/nite/fd2log&
```

To restart **runacct** at a specific state:

```
nohup runacct 0601 WTMPFIX  2>  /usr/adm/acct/nite/fd2log&
```

### FIXING CORRUPTED FILES

Unfortunately, this accounting system is not entirely fool proof. Occasionally, a file will become corrupted or lost. Some of the files can simply be ignored or restored from the file save backup. However, certain files must be fixed in order to maintain the integrity of the accounting system.

### A. Fixing WTMP Errors

The *wtmp* files seem to cause the most problems in the day to day operation of the accounting system. When the date is changed and the UNIX system is in multiuser mode, a set of date change records is written into */usr/adm/wtmp*. The **wtmpfix** program is designed to adjust the time stamps in the *wtmp* records when a date change is encountered. Some combinations of date changes and reboots, however, will slip through **wtmpfix** and cause **acctcon1** to fail. The following steps show how to patch up a *wtmp* file.

```
cd /usr/adm/acct/nite
fwtmp < wtmp.MMDD > xwtmp
ed xwtmp
   delete corrupted records or
   delete all records from beginning up to the date change
fwtmp -ic < xwtmp > wtmp.MMDD
```

If the *wtmp* file is beyond repair, create a null *wtmp* file. This will prevent any charging of connect time. **Acctprc1** will not be able to determine which login owned a particular process, but it will be charged to the login that is first in the password file for that user id.

### B. Fixing TACCT Errors

If the installation is using the accounting system to charge users for system resources, the integrity of *sum/tacct* is quite important. Occasionally, mysterious *tacct* records will appear with negative numbers, duplicate user IDs, or a user ID of 65,535. First check *sum/tacctprev* with **prtacct**. If it looks all right, the latest *sum/tacct.MMDD* should be patched up, then *sum/tacct* recreated. A simple patchup procedure would be:

```
cd /usr/adm/acct/sum
acctmerg -v < tacct.MMDD > xtacct
ed xtacct
   remove the bad records
   write duplicate uid records to another file
acctmerg -i < xtacct > tacct.MMDD
acctmerg tacctprev < tacct.MMDD > tacct
```

Remember that the **monacct** procedure removes all the *tacct.MMDD* files; therefore, *sum/tacct* can be recreated by merging these files together.

## UPDATING PNPSPLIT

The **pnpsplit** subroutine is used by **acctcon1** and **acctprc1** to determine the difference between prime and nonprime time. Prime time is defaulted from 9:00 am to 5:00 pm, Monday through Friday. Nonprime time is considered to be all other hours and the entire day for those days listed in the **holidays** structure in *pnpsplit.c*. The holidays listed are accurate for Bell Laboratories New Jersey locations for the year the operating system was released. Every year on the day after Christmas (the last holiday of the calendar year), the following message will be printed on the system console terminal and appear in *log*.

```
*** RECOMPILE pnpsplit WITH NEW HOLIDAYS ***
```

This message will continue to be sent each time the accounting is run until **pnpsplit**, **acctcon1**, and **acctprc1** are recompiled. The following steps should be taken to successfully recompile these programs.

1. Edit *pnpsplit.c* to change the *thisyear* variable to the new year. Update the *holidays* structure to reflect the new holidays. The numeric entry in the structure is the day of the year, less one. For example, New Year's Day (January 1) is entered as 0. *Pnpsplit.c* is in */usr/src/cmd/acct/lib*.

2. Update the accounting library **a.a** and recompile **acctprc1**, and **acctcon1** by:

superuser to root

ARGS= " acctcon1 acctprc1 " /usr/src/:mkcmd acct

**DAILY REPORTS**

**Runacct** generates five basic reports upon each invocation. Samples of these reports are shown in Attachment 7.2. They cover the areas of connect accounting, usage by person on a daily basis, command usage reported by daily and monthly totals, and a report of the last time users were logged in.

The following paragraphs describe the reports and the meanings of their tabulated data.

**A. Daily Report**

In the first part of the report, the **from/to** banner should alert the administrator to the period reported on. The times are the time the last accounting report was generated until the time the current accounting report was generated. It is followed by a log of system reboots, shutdowns, power fail recoveries, and any other record dumped into */usr/adm/wtmp* by the **acctwtmp** program [see **acct**(1M) in the UNIX System Administrator's Manual].

The second part of the report is a breakdown of line utilization. The TOTAL DURATION tells how long the system was in multiuser state (able to be accessed through the terminal lines). The columns are:

| | |
|---|---|
| LINE | The terminal line or access port. |
| MINUTES | The total number of minutes that line was in use during the accounting period. |
| PERCENT | The total number of MINUTES the line was in use divided into the TOTAL DURATION. |
| # SESS | The number of times this port was accessed for a **login**(1) session. |
| # ON | This column does not have much meaning anymore. It used to give the number of times that the port was used to log a user on; but since **login**(1) can no longer be executed explicitly to log a new user in, this column should be identical with SESS. |
| # OFF | This column reflects not just the number of times a user logged off but also any interrupts that occur on that line. Generally, interrupts occur on a port when the **getty**(8) is first invoked when the system is brought to multiuser state. These interrupts occur at a rate of about two per event; therefore, it is not uncommon to see in excess of twice the amount of OFF than ON or SESS. Where this column does come into play is when the # OFF exceeds the # ON by a large factor. This usually indicates that the multiplexer, modem or cable is going bad, or there is a bad connection somewhere. The most common cause of this is an unconnected cable dangling from the multiplexer. |

During real time, */usr/adm/wtmp* should be monitored as this is the file that the connect accounting is geared from. If it grows rapidly, execute **acctcon1** to see which tty line is the most noisy. If the interrupting is occurring at a furious rate, general system performance will be effected.

**B. Daily Usage Report**

This report gives a by-user breakdown of system resource utilization. Its data consists of:

| | |
|---|---|
| UID | The user ID. |

LOGIN NAME — The login name of the user; there can be more than one login name for a single user ID, this identifies which one.

CPU (MINS) — This represents the amount of time the user's process used the central processing unit. This category is broken down into PRIME and NPRIME (nonprime) utilization. The accounting system's idea of this breakdown is located in the accounting library function **pnpsplit** where the **holidays** array, which also determines nonprime time, is also defined. As delivered, prime time is defined to be 0900-1700 hours. The **holidays** array is correct for Bell Laboratories New Jersey locations for the year of the release.

KCORE-MINS — This represents a cummulative measure of the amount of memory a process uses while running. The amount shown reflects kilobyte segments of memory used per minute. This measurement is also broken down into PRIME and NPRIME amounts.

CONNECT (MINS) — This identifies "Real Time" used. What this column really identifies is the amount of time that a user was logged into the system. If this time is rather high and the later column called # OF PROCS is low, this user is what is called a "line hog". That is, this person logs in first thing in the morning and does not hardly touch the terminal the rest of the day. Watch out for these kind of users. This column is also subdivided into PRIME and NPRIME utilization.

DISK BLOCKS — When the disk accounting programs have been run, their output is merged into the total accounting record (*tacct.h*) and shows up in this column. This disk accounting is accomplished by the program **acctdusg**.

# OF PROCS — This column reflects the number of processes that was invoked by the user. This is a good column to watch for large numbers indicating that a user may have a shell procedure that runs amock. The most common example of this is for a **crontab** entry to try to execute a user's *.profiie* via **su-** that unfortunately prompts for a terminal type and sits in an endless loop trying to read from the terminal (there is not one when **cron** is executing a process). Preventive coding is encouraged in the *.profile*.

# OF SESS — This is how many times the user logged onto the system.

# DISK SAMPLES — This indicates how many times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.

FEE — An often unused field in the total accounting record, the FEE represents the total accumulation of widgets charged against the user by the **chargefee** shell procedure [see **acctsh**(1M)]. The **chargefee** procedure is used to levy charges against a user for special services performed such as file restores, tape manipulation by operators, etc.

### C. Daily Command and Monthly Total Command Summaries

These two reports are virtually the same except that the Daily Command Summary only reports on the current accounting period while the Monthly Total Command Summary tells the story for the start of the fiscal period to the current date. In other words, the monthly report reflects the data accumulated since the last invocation of **monacct**.

The data included in these reports gives an administrator an idea as to the heaviest used commands; and based on those commands' characteristics of system resource utilization, a hint as to what to weigh more heavily when system tuning.

These reports are sorted by TOTAL KCOREMIN which is an arbitrary yardstick, but often a good one for calculating "drain" on a system.

COMMAND NAME This is the name of the command. Unfortunately, all shell procedures are lumped together under the name **sh** since only object modules are reported by the process accounting system. The administrator should monitor the frequency of programs called **a.out** or **core** or any other name that does not seem quite right. Often people like to work on their favorite version of backgammon only they do not want everyone to know about it. **Acctcom** is also a good tool to use for determining who executed a suspiciously named command and also if superuser privileges were used.

NUMBER CMDS This is the total number of invocations of this particular command.

TOTAL KCOREMIN The total cummulative measurement of the amount of kilobyte segments of memory used by a process per minute of run time.

TOTAL CPU-MIN The total processing time this program has accumulated.

TOTAL REAL-MIN The total real-time (wall-clock) minutes this program has accumulated. This total is the actual "waited for" time as opposed to kicking off a process in the background.

MEAN SIZE-K This is the mean of the TOTAL KCOREMIN over the number of invocations reflected by NUMBER CMDS.

MEAN CPU-MIN This is the mean derived between the NUMBER CMDS and TOTAL CPU-MIN.

HOG FACTOR This is a relative measurement of the ratio of system availability to system utilization. It is computed by the formula

(total CPU time) / (elapsed time)

This gives a relative measure of the total available CPU time consumed by the process during its execution.

CHARS TRNSFD This column, which may go negative, is a total count of the number of characters pushed around by the **read**(2) and **write**(2) system calls.

BLOCKS READ A total count of the physical block reads and writes that a process performed.

### D. Last Login

This report simply gives the date when a particular login was last used. This could be a good source for finding likely candidates for the tape archives or getting rid of unused logins and login directories.

### SUMMARY

The UNIX System Accounting was designed from a UNIX system administrator's point of view. Every possible precaution has been taken to ensure that the system will run smoothly and without error. It is important to become familiar with the C programs and shell procedures. The manual pages should be studied, and it is advisable to keep a printed copy of the shell procedures handy. The accounting system should be easy to maintain, provide valuable information for the administrator, and provide accurate breakdowns of the usage of system resources for charging purposes.

## ATTACHMENT 7.1

**Format of wtmp files (utmp.h):**

```
/*      %W%      */
/*      <sys/types.h> must be included.                                  */
#define UTMP_FILE       "/etc/utmp"
#define WTMP_FILE       "/etc/wtmp"
#define ut_name ut_user

struct  utmp
  {
        char ut_user[8] ;               /* User login name */
        char ut_id[4] ;                 /* /etc/lines id(usually line #) */
        char ut_line[12] ;              /* device name (console, lnxx) */
        short ut_pid ;                  /* process id */
        short ut_type ;                 /* type of entry */
        struct exit_status
          {
            short e_termination ;       /* Process termination status */
            short e_exit ;              /* Process exit status */
          }
        ut_exit ;                       /* The exit status of a process
                                         * marked as DEAD_PROCESS.
                                         */
        time_t ut_time ;                /* time entry was made */

    } ;
/*      Definitions for ut_type                                          */

#define EMPTY           0
#define RUN_LVL         1
#define BOOT_TIME       2
#define OLD_TIME        3
#define NEW_TIME        4
#define INIT_PROCESS    5       /* Process spawned by "init" */
#define LOGIN_PROCESS   6       /* A "getty" process waiting for login */
#define USER_PROCESS    7       /* A user process */
#define DEAD_PROCESS    8
#define ACCOUNTING      9

#define UTMAXTYPE       ACCOUNTING      /* Largest legal value of ut_type */

/*      Special strings or formats used in the "ut_line" field when      */
/*      accounting for something other than a process.                   */
/*      No string for the ut_line field can be more than 11 chars +      */
/*      a NULL in length.                                                */

#define RUNLVL_MSG      "run-level %c"
#define BOOT_MSG        "system boot"
#define OTIME_MSG       "old time"
#define NTIME_MSG       "new time"
```

ATTACHMENT 7.1 (Contd)

**Definitions (acctdef.h):**

```
/*      %W% of %G%      */
/*
 *      defines, typedefs, etc. used by acct programs
 */


/*
 *      acct only typedefs
 */
typedef unsigned short  uid_t;

#ifdef u3b
#define HZ      100
#else
#define HZ      60
#endif

#define LSZ     12      /* sizeof line name */
#define NSZ     8       /* sizeof login name */
#define P       0       /* prime time */
#define NP      1       /* nonprime time */

/*
 *      limits which may have to be increased if systems get larger
 */
#define SSIZE   1000    /* max number of sessions in 1 acct run */
#define TSIZE   100     /* max number of line names in 1 acct run */
#define USIZE   500     /* max number of distinct login names in 1 acct run */

#define EQN(s1, s2)     (strncmp(s1, s2, sizeof(s1)) == 0)
#define CPYN(s1, s2)    strncpy(s1, s2, sizeof(s1))

#define SECSINDAY       86400L
#define SECS(tics)      ((double) tics)/HZ
#define MINS(secs)      ((double) secs)/60
#define MINT(tics)      ((double) tics)/(60*HZ)

#ifdef pdp11
#define KCORE(clicks)   ((double) clicks/16)
#endif
#ifdef vax
#define KCORE(clicks)   ((double) clicks/2)
#endif
#ifdef u3b
#define KCORE(clicks)   ((double) clicks*2)
#endif
```

ATTACHMENT 7.1 (Contd)

**Format of pacct files (acct.h):**

```
/*
 * Accounting structures
 */

typedef ushort comp_t;          /* "floating point" */
                /* 13-bit fraction, 3-bit exponent */

struct  acct

        char    ac_flag;        /*Accounting flag */
        char    ac_stat;        /*Exit status */
        ushort  ac_uid;         /*Accounting user ID */
        ushort  ac_gid;         /*Accounting group ID */
        dev_t   ac_tty;         /*control typewriter */
        time_t  ac_btime;       /*Beginning time */
        comp_t  ac_utime;       /*acctng user time in clock ticks */
        comp_t  ac_stime;       /*acctng system time in clock ticks */
        comp_t  ac_etime;       /*acctng elapsed time in clock ticks */
        comp_t  ac_mem;         /*memory usage */
        comp_t  ac_io;          /*chars transferred */
        comp_t  ac_rw;          /*blocks read or written */
        char    ac_comm[8];     /*command name */
 ;

extern  struct  acct    acctbuf;
extern  struct  inode   *acctp; /*inode of accounting file */

#defineAFORK            01      /*has executed fork, but no exec */
#defineASU              02      /*used superuser privileges */
#defineACCTF            0300    /*record type: 00 = acct */
```

**Format of tacct files (tacct.h):**

```
/*
 *      total acctounting (for acct period), also for day
 */

struct  tacct
        uid_t           ta_uid;         /*userid */
        char            ta_name[8];     /*login name */
        float           ta_cpu[2];      /*cum. cpu time, p/np (mins) */
        float           ta_kcore[2];    /*cum kcore-minutes, p/np */
        float           ta_con[2];      /*cum. connect time, p/np, mins */
        float           ta_du;          /*cum. disk usage */
        long            ta_pc;          /*count of processes */
        unsigned short  ta_sc;          /*count of login sessions */
        unsigned short  ta_dc;          /*count of disk samples */
        unsigned short  ta_fee;         /*fee for special services */
 ;
```

ATTACHMENT 7.1 (Contd)

Format of ctmp file (ctmp.h):

```
/*
 *      connect time record (various intermediate files)
 */
struct ctmp
        dev_t   ct_tty;         /*major minor */
        uid_t   ct_uid;         /*userid */
        char    ct_name[8];     /*login name */
        long    ct_con[2];      /*connect time (p/np) secs */
        time_t  ct_start;       /*session start time */
;
```

ATTACHMENT 7.2

Jun 8 04:14 1979 DAILY REPORT FOR pwba Page 1

from Thu Jun 7 06:00:48 1979

to Fri Jun 8 04:00:28 1979

2 shutdown

2 pwa

TOTAL DRATION IS 1320 MINUTES

| LINE | MINUTES | PERCENT | # SESS | # ON | # OFF |
|---|---|---|---|---|---|
| tty04 | 479 | 36 | 9 | 9 | 30 |
| tty47 | 341 | 26 | 4 | 4 | 33 |
| tty44 | 298 | 23 | 3 | 3 | 29 |
| tty46 | 336 | 25 | 9 | 9 | 33 |
| console | 1100 | 83 | 14 | 14 | 21 |
| tty05 | 448 | 34 | 3 | 3 | 22 |
| tty06 | 439 | 33 | 9 | 9 | 31 |
| tty07 | 421 | 32 | 6 | 6 | 24 |
| tty42 | 53 | 4 | 5 | 5 | 20 |
| tty09 | 385 | 29 | 11 | 11 | 33 |
| tty10 | 336 | 25 | 10 | 10 | 31 |
| tty08 | 464 | 35 | 2 | 2 | 19 |
| tty26 | 544 | 41 | 6 | 6 | 24 |
| tty12 | 252 | 19 | 5 | 5 | 25 |
| tty13 | 258 | 20 | 3 | 3 | 21 |
| tty14 | 156 | 12 | 6 | 6 | 26 |
| tty17 | 145 | 11 | 1 | 1 | 16 |
| tty18 | 39 | 3 | 5 | 5 | 24 |
| tty15 | 228 | 17 | 5 | 5 | 25 |
| tty25 | 704 | 53 | 6 | 6 | 25 |
| tty21 | 0 | 0 | 0 | 0 | 16 |
| tty19 | 10 | 1 | 1 | 1 | 17 |
| tty20 | 25 | 2 | 2 | 2 | 18 |
| tty22 | 0 | 0 | 0 | 0 | 15 |
| tty23 | 0 | 0 | 0 | 0 | 15 |
| tty24 | 0 | 0 | 0 | 0 | 16 |
| tty27 | 481 | 36 | 3 | 3 | 20 |
| tty28 | 426 | 32 | 5 | 5 | 24 |
| tty29 | 302 | 23 | 6 | 6 | 25 |
| tty30 | 257 | 20 | 11 | 11 | 28 |
| tty40 | 380 | 29 | 5 | 5 | 21 |
| tty41 | 343 | 26 | 3 | 3 | 21 |
| tty45 | 0 | 0 | 0 | 0 | 15 |
| tty11 | 365 | 28 | 7 | 7 | 25 |
| tty43 | 3 | 0 | 1 | 1 | 17 |
| tty16 | 213 | 16 | 3 | 3 | 20 |
| tty31 | 250 | 19 | 4 | 4 | 18 |
| tty02 | 62 | 5 | 1 | 1 | 3 |
| TOTALS | 10544 | -- | 174 | 174 | 846 |

ATTACHMENT 7.2 (Contd)

Jun 8 04:14 1979 DAILY USAGE REPORT FOR pwba Page 1

| UID | LOGIN NAME | CPU (MINS) PRIME | CPU (MINS) NPRIME | KCORE-MINS PRIME | KCORE-MINS NPRIME | CONNECT (MINS) PRIME | CONNECT (MINS) NPRIME | DISK BLOCKS | # OF PROCS | # OF SESS | # DISK SAMPLES | FEE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | TOTAL | 388 | 103 | 12414 | 2934 | 9251 | 1056 | 0 | 16164 | 174 | 0 | 0 |
| 0 | root | 47 | 41 | 1003 | 924 | 67 | 30 | 0 | 2360 | 8 | 0 | 0 |
| 4 | adm | 27 | 19 | 48 | 652 | 0 | 0 | 0 | 842 | 0 | 0 | 0 |
| 19 | games | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 28 | 0 | 0 | 0 |
| 22 | mhb | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 14 | 2 | 0 | 0 |
| 37 | abs | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
| 37 | abajrk | 14 | 0 | 284 | 0 | 423 | 0 | 0 | 1588 | 4 | 0 | 0 |
| 68 | rje | 3 | 3 | 24 | 21 | 0 | 0 | 0 | 179 | 0 | 0 | 0 |
| 71 | ? | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 |
| 150 | jac | 7 | 0 | 156 | 5 | 281 | 2 | 0 | 510 | 13 | 0 | 0 |
| 173 | ? | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 0 |
| 180 | ? | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 185 | ? | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| 217 | denise | 0 | 0 | 2 | 0 | 31 | 0 | 0 | 32 | 3 | 0 | 0 |
| 217 | kof | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 7 | 1 | 0 | 0 |
| 219 | ? | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 |
| 1001 | hsm | 5 | 0 | 189 | 0 | 179 | 0 | 0 | 92 | 2 | 0 | 0 |
| 2001 | systst | 0 | 1 | 5 | 28 | 476 | 64 | 0 | 99 | 5 | 0 | 0 |
| 2002 | mfp | 1 | 0 | 7 | 5 | 270 | 62 | 0 | 93 | 3 | 0 | 0 |
| 2003 | als | 1 | 0 | 23 | 0 | 100 | 0 | 0 | 99 | 3 | 0 | 0 |
| 2005 | eric | 0 | 0 | 3 | 0 | 13 | 0 | 0 | 21 | 1 | 0 | 0 |
| 2006 | hoot | 0 | 0 | 2 | 0 | 16 | 0 | 0 | 8 | 1 | 0 | 0 |
| 2009 | agp | 47 | 0 | 2040 | 0 | 444 | 0 | 0 | 492 | 2 | 0 | 0 |
| 2009 | fsrepl | 2 | 0 | 60 | 0 | 36 | 0 | 0 | 96 | 1 | 0 | 0 |
| 2011 | pdw | 0 | 0 | 1 | 0 | 4 | 0 | 0 | 11 | 1 | 0 | 0 |
| 2012 | pwbst | 0 | 0 | 1 | 0 | 28 | 0 | 0 | 9 | 1 | 0 | 0 |
| 2014 | cath | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 7 | 1 | 0 | 0 |
| 2022 | rem | 32 | 1 | 1227 | 91 | 576 | 4 | 0 | 226 | 3 | 0 | 0 |
| 2025 | fld | 55 | 23 | 2176 | 862 | 336 | 98 | 0 | 750 | 7 | 0 | 0 |
| 2027 | krb | 14 | 2 | 365 | 51 | 547 | 24 | 0 | 372 | 8 | 0 | 0 |
| 2028 | text | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 13 | 1 | 0 | 0 |
| 2030 | arf | 8 | 0 | 288 | 0 | 317 | 0 | 0 | 315 | 3 | 0 | 0 |
| 2031 | dp | 12 | 0 | 480 | 3 | 459 | 6 | 0 | 220 | 6 | 0 | 0 |
| 2032 | graf | 2 | 0 | 49 | 0 | 23 | 0 | 0 | 118 | 1 | 0 | 0 |
| 2033 | ecp | 3 | 0 | 74 | 0 | 355 | 0 | 0 | 115 | 4 | 0 | 0 |
| 2040 | leap | 15 | 0 | 308 | 0 | 513 | 1 | 0 | 505 | 2 | 0 | 0 |
| 2041 | dan | 3 | 0 | 93 | 3 | 149 | 2 | 0 | 117 | 8 | 0 | 0 |
| 2051 | da52 | 2 | 2 | 19 | 40 | 375 | 601 | 0 | 611 | 8 | 0 | 0 |
| 2055 | nuucp | 0 | 0 | 15 | 9 | 17 | 1 | 0 | 10 | 3 | 0 | 0 |
| 2057 | ech | 1 | 0 | 28 | 0 | 63 | 0 | 0 | 68 | 2 | 0 | 0 |
| 2061 | jcw | 4 | 3 | 99 | 70 | 37 | 34 | 0 | 869 | 4 | 0 | 0 |
| 2064 | mjr | 18 | 0 | 443 | 0 | 176 | 0 | 0 | 2065 | 3 | 0 | 0 |
| 2065 | rrr | 0 | 0 | 6 | 0 | 7 | 0 | 0 | 23 | 1 | 0 | 0 |
| 2068 | trc | 0 | 0 | 7 | 0 | 10 | 0 | 0 | 29 | 1 | 0 | 0 |
| 2075 | herb | 29 | 0 | 1178 | 1 | 384 | 2 | 0 | 249 | 5 | 0 | 0 |
| 2086 | paul | 1 | 0 | 14 | 0 | 152 | 0 | 0 | 28 | 1 | 0 | 0 |
| 2087 | pris | 0 | 0 | 0 | 10 | 0 | 2 | 0 | 13 | 1 | 0 | 0 |
| 2111 | pwvcs | 2 | 3 | 60 | 85 | 64 | 86 | 0 | 185 | 4 | 0 | 0 |
| 2116 | rbj | 1 | 0 | 16 | 0 | 408 | 0 | 0 | 222 | 1 | 0 | 0 |
| 2121 | teach | 0 | 0 | 3 | 0 | 53 | 0 | 0 | 50 | 2 | 0 | 0 |
| 2123 | msb | 0 | 0 | 3 | 0 | 5 | 0 | 0 | 24 | 1 | 0 | 0 |
| 2124 | rnt | 2 | 0 | 42 | 0 | 66 | 0 | 0 | 260 | 3 | 0 | 0 |
| 2126 | dal | 0 | 0 | 5 | 0 | 121 | 0 | 0 | 17 | 1 | 0 | 0 |
| 2127 | m2 | 15 | 0 | 495 | 11 | 390 | 2 | 0 | 602 | 10 | 0 | 0 |

Jun 8 04:14 1979 DAILY USAGE REPORT FOR pwba Page 2

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2128 | jel | 14 | 0 | 492 | 9 | 422 | 14 | 0 | 523 | 8 | 0 | 0 |
| 2130 | sl | 0 | 0 | 5 | 1 | 16 | 0 | 0 | 42 | 2 | 0 | 0 |
| 2130 | s3 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 9 | 1 | 0 | 0 |
| 2135 | jfn | 0 | 1 | 0 | 12 | 0 | 11 | 0 | 33 | 2 | 0 | 0 |
| 2136 | m2class | 0 | 0 | 5 | 0 | 2 | 0 | 0 | 18 | 1 | 0 | 0 |
| 2140 | star | 4 | 0 | 213 | 12 | 90 | 3 | 0 | 170 | 7 | 0 | 0 |
| 2141 | reg | 5 | 0 | 245 | 25 | 470 | 4 | 0 | 181 | 1 | 0 | 0 |
| 2199 | llc | 0 | 0 | 1 | 0 | 10 | 0 | 0 | 7 | 1 | 0 | 0 |
| 2999 | stock | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 17 | 1 | 0 | 0 |
| 3001 | whm | 5 | 0 | 93 | 0 | 253 | 0 | 0 | 414 | 3 | 0 | 0 |
| 3332 | vjf | 0 | 0 | 4 | 0 | 8 | 0 | 0 | 39 | 1 | 0 | 0 |

ATTACHMENT 7.2 (Contd)

Jun 8 04:07 1979 DAILY COMMAND SUMMARY Page 1

| COMMAND NAME | NUMBER CMDS | TOTAL KCOREMIN | TOTAL CPU-MIN | TOTAL REAL-MIN | MEAN SIZE-K | MEAN CPU-MIN | HOG FACTOR | CHARS TRNSFD | BLOCKS READ |
|---|---|---|---|---|---|---|---|---|---|
| TOTALS | 16164 | 15332.89 | 490.72 | 37463.98 | 31.25 | 003 | 0.01 | 322183844 | 1097670 |
| nroff | 119 | 3958.68 | 93.21 | 569.83 | 42.47 | 0.78 | 0.16 | 67070052 | 130284 |
| troff | 26 | 2483.38 | 51.63 | 342.70 | 48.10 | 1.99 | 0.15 | 37869304 | 48939 |
| xnroff | 20 | 732.03 | 16.74 | 111.05 | 43.73 | 0.84 | 0.15 | 13385248 | 22659 |
| a.out | 31 | 623.53 | 10.52 | 142.77 | 59.26 | 0.34 | 0.07 | 382435 | 2758 |
| egrep | 185 | 574.83 | 13.96 | 34.53 | 41.18 | 0.08 | 0.40 | 170625 | 8249 |
| m2fins | 232 | 555.79 | 9.93 | 155.11 | 55.96 | 0.04 | 0.06 | 6155937 | 30994 |
| c1 | 150 | 519.04 | 13.57 | 48.89 | 38.25 | 0.09 | 0.28 | 4285724 | 16032 |
| c0 | 165 | 413.10 | 9.19 | 35.16 | 44.93 | 0.06 | 0.26 | 3527309 | 12170 |
| m2edit | 33 | 340.92 | 4.63 | 148.27 | 73.62 | 0.14 | 0.03 | 1074914 | 14492 |
| ld | 87 | 317.38 | 7.94 | 38.48 | 39.97 | 0.09 | 0.21 | 17610896 | 45797 |
| acctcms | 17 | 294.75 | 6.49 | 14.15 | 45.41 | 0.38 | 0.46 | 2525427 | 5515 |
| c2 | 112 | 289.69 | 9.13 | 34.61 | 31.72 | 0.08 | 0.26 | 3667050 | 9681 |
| sh | 1634 | 276.98 | 26.77 | 20444.24 | 10.35 | 0.01 | 0.00 | 3496613 | 71979 |
| ed | 524 | 253.13 | 14.46 | 2029.89 | 17.50 | 0.03 | 0.01 | 18058108 | 56039 |
| acctprc1 | 3 | 231.28 | 6.67 | 19.45 | 34.67 | 2.22 | 0.34 | 2577344 | 2926 |
| du | 145 | 219.35 | 19.91 | 39.08 | 11.02 | 0.14 | 0.51 | 716389 | 23695 |
| diff | 49 | 175.53 | 6.04 | 25.78 | 29.05 | 0.12 | 0.23 | 3740887 | 11351 |
| get | 151 | 152.96 | 4.28 | 25.23 | 35.74 | 0.03 | 0.17 | 3634042 | 24917 |
| adb | 22 | 148.10 | 4.07 | 202.35 | 36.37 | 0.19 | 0.02 | 2313718 | 9813 |
| tbl | 24 | 143.43 | 2.44 | 210.65 | 58.71 | 0.10 | 0.01 | 1536210 | 3433 |
| dd | 9 | 139.24 | 10.15 | 51.05 | 13.72 | 1.13 | 0.20 | 26006848 | 294 |
| as2 | 155 | 120.33 | 9.82 | 42.25 | 12.17 | 0.06 | 0.23 | 10500835 | 30165 |
| sed | 597 | 115.46 | 4.19 | 36.23 | 27.57 | 0.01 | 0.12 | 783825 | 24497 |
| ps | 51 | 109.69 | 5.92 | 41.55 | 18.54 | 0.12 | 0.14 | 2278056 | 8310 |
| make | 89 | 102.94 | 2.87 | 203.32 | 35.81 | 0.03 | 0.01 | 1018461 | 8664 |
| delta | 25 | 90.23 | 2.27 | 17.80 | 39.70 | 0.09 | 0.13 | 2909269 | 9321 |
| cpp | 172 | 89.37 | 2.69 | 11.22 | 33.19 | 0.02 | 0.24 | 3519054 | 12155 |
| fsck | 16 | 86.94 | 1.30 | 10.57 | 66.85 | 0.08 | 0.12 | 27671849 | 2927 |
| find | 52 | 86.64 | 5.05 | 63.87 | 17.15 | 0.10 | 0.08 | 565125 | 11161 |
| ls | 706 | 82.47 | 5.78 | 62.85 | 14.26 | 0.01 | 0.09 | 1811882 | 29659 |
| xck | 2 | 79.44 | 10.49 | 47.89 | 7.57 | 5.25 | 0.22 | 198016 | 21935 |
| awk | 22 | 78.83 | 1.37 | 5.24 | 57.72 | 0.06 | 0.26 | 355466 | 3769 |
| uucico | 60 | 75.55 | 1.42 | 632.50 | 53.27 | 0.02 | 0.00 | 398693 | 6377 |
| acctcom | 9 | 75.21 | 2.81 | 11.49 | 26.75 | 0.31 | 0.24 | 1283776 | 3771 |
| echo | 2814 | 66.10 | 7.08 | 91.80 | 9.33 | 0.00 | 0.08 | 168651 | 24253 |
| ged | 3 | 57.27 | 0.82 | 7.51 | 70.16 | 0.27 | 0.11 | 51832 | 426 |
| dc | 284 | 56.92 | 2.42 | 9.43 | 23.48 | 0.01 | 0.26 | 14253 | 20329 |
| 450 | 7 | 48.03 | 6.80 | 84.45 | 7.06 | 0.97 | 0.08 | 279451 | 1700 |
| cat | 749 | 45.49 | 5.69 | 478.54 | 8.00 | 0.01 | 0.01 | 8959500 | 27903 |
| ntd | 6 | 41.52 | 1.55 | 7.55 | 26.87 | 0.26 | 0.20 | 59688 | 478 |
| mail | 202 | 39.95 | 2.05 | 532.98 | 19.53 | 0.01 | 0.00 | 427217 | 14377 |
| acctprc2 | 3 | 38.95 | 1.43 | 19.45 | 27.24 | 0.48 | 0.07 | 587336 | 87 |
| sort | 94 | 38.72 | 1.09 | 9.73 | 35.41 | 0.01 | 0.11 | 375876 | 4433 |
| pr | 104 | 34.89 | 2.47 | 214.50 | 14.10 | 0.02 | 0.01 | 1060989 | 6572 |
| haspmain | 7 | 33.20 | 5.28 | 1244.54 | 6.29 | 0.75 | 0.00 | 63064 | 36635 |
| ex | 17 | 31.69 | 0.62 | 41.04 | 50.97 | 0.04 | 0.02 | 514624 | 3593 |
| grep | 213 | 28.73 | 2.98 | 21.01 | 9.64 | 0.01 | 0.14 | 2100229 | 14297 |

**ATTACHMENT 7.2 (Contd)**

Jun 8 04:07 1979 MONTHLY TOTAL SUMMARY Page 1

| COMMAND NAME | NUMBER CMDS | TOTAL KCOREMIN | TOTAL CPU-MIN | TOTAL REAL-MIN | MEAN SIZE-K | MEAN CPU-MIN | HOG FACTOR | CHARS TRNSFD | BLOCKS READ |
|---|---|---|---|---|---|---|---|---|---|
| TOTALS | 553286 | 297698.78 | 10916.09 | 742924.94 | 27.27 | 0.02 | 0.01 | 820472546 | 26253312 |
| nroff | 1687 | 44681.55 | 995.92 | 5737.25 | 44.86 | 0.59 | 0.17 | 613403153 | 1089180 |
| troff | 1351 | 25692.15 | 583.69 | 4356.05 | 44.02 | 0.43 | 0.13 | 413163589 | 646243 |
| spellpro | 6466 | 17296.41 | 294.16 | 1893.79 | 58.81 | 0.05 | 0.16 | 334572640 | 853901 |
| m2edit | 654 | 13526.69 | 164.62 | 4238.58 | 82.17 | 0.25 | 0.04 | 54940426 | 427924 |
| xnroff | 397 | 10408.44 | 203.72 | 1496.32 | 51.09 | 0.51 | 0.14 | 215221419 | 301967 |
| sort | 7963 | 9292.34 | 226.01 | 2298.05 | 41.11 | 0.03 | 0.10 | 80108304 | 355963 |
| cl | 6139 | 8949.86 | 236.45 | 861.09 | 37.85 | 0.04 | 0.27 | 79897995 | 489661 |
| ld | 3244 | 8852.96 | 223.19 | 1128.09 | 39.67 | 0.07 | 0.20 | 493701995 | 1278119 |
| sed | 53134 | 8126.71 | 313.85 | 2241.78 | 25.89 | 0.01 | 0.14 | 23035033 | 1692990 |
| m2find | 2982 | 7984.45 | 140.18 | 1698.25 | 56.96 | 0.05 | 0.08 | 111330040 | 449604 |
| c0 | 6586 | 7866.42 | 185.16 | 725.47 | 42.49 | 0.03 | 0.26 | 72595655 | 389426 |
| ed | 20083 | 7822.78 | 425.90 | 41898.18 | 18.37 | 0.02 | 0.01 | 483425634 | 1541326 |
| tbl | 660 | 7766.69 | 113.95 | 2458.55 | 68.16 | 0.17 | 0.05 | 50760094 | 83887 |
| sh | 40476 | 7499.67 | 635.00 | 383786.53 | 11.81 | 0.02 | 0.00 | 70525236 | 1421194 |
| du | 1941 | 6730.54 | 553.04 | 1128.44 | 12.17 | 0.28 | 0.49 | 20848359 | 628324 |
| a.out | 1483 | 5658.46 | 126.87 | 1868.87 | 44.60 | 0.09 | 0.07 | 16158675 | 80260 |
| egrep | 4801 | 5573.51 | 139.86 | 460.25 | 39.85 | 0.03 | 0.30 | 6823696 | 237208 |
| lint1 | 793 | 5325.66 | 71.23 | 425.67 | 74.76 | 0.09 | 0.17 | 9599001 | 131592 |
| cat | 21170 | 4657.53 | 236.59 | 4354.24 | 19.69 | 0.01 | 0.05 | 239180412 | 1023965 |
| acctprc1 | 42 | 3837.84 | 110.88 | 291.34 | 34.61 | 2.64 | 0.38 | 43954136 | 61123 |
| c2 | 4067 | 3807.25 | 144.86 | 477.28 | 26.28 | 0.04 | 0.30 | 57519376 | 213521 |
| grep | 21212 | 3204.86 | 300.44 | 2727.87 | 10.67 | 0.01 | 0.11 | 139340583 | 899415 |
| cpp | 7469 | 3060.72 | 94.12 | 647.79 | 32.52 | 0.01 | 0.15 | 91471956 | 459882 |
| getty | 35556 | 2948.71 | 853.53 | 101107.45 | 3.45 | 0.02 | 0.01 | 34704751 | 263866 |
| m2editD | 83 | 2707.27 | 28.79 | 361.84 | 94.02 | 0.35 | 0.08 | 2852202 | 33949 |
| as2 | 6454 | 2698.74 | 218.96 | 910.59 | 12.33 | 0.03 | 0.24 | 213336016 | 705690 |
| make | 1858 | 2449.10 | 64.69 | 4388.86 | 37.86 | 0.03 | 0.01 | 24116259 | 175544 |
| ps | 1034 | 2384.14 | 128.29 | 1207.87 | 18.58 | 0.12 | 0.11 | 54873792 | 204172 |
| acctcms | 294 | 2288.36 | 51.99 | 116.06 | 44.01 | 0.18 | 0.45 | 36124940 | 80523 |
| uucico | 815 | 2226.75 | 40.42 | 11729.01 | 55.08 | 0.05 | 0.00 | 11086105 | 162558 |
| ls | 18876 | 2170.01 | 152.76 | 1538.09 | 14.20 | 0.01 | 0.10 | 32418106 | 691028 |
| find | 1705 | 2114.18 | 114.35 | 920.75 | 18.49 | 0.07 | 0.12 | 94631199 | 338600 |
| ged | 72 | 2026.43 | 28.54 | 317.21 | 71.01 | 0.40 | 0.09 | 1648636 | 10374 |
| echo | 84710 | 2018.23 | 190.14 | 1138.49 | 10.61 | 0.00 | 0.17 | 2926992 | 649200 |
| cpio | 127 | 1956.60 | 77.03 | 391.45 | 25.40 | 0.61 | 0.20 | 190822346 | 296302 |
| maze | 8 | 1620.42 | 44.80 | 128.25 | 36.17 | 5.60 | 0.35 | 120399 | 212 |
| mail | 4735 | 1474.38 | 76.92 | 14262.62 | 19.17 | 0.02 | 0.01 | 25719618 | 463748 |
| get | 1085 | 1358.03 | 37.59 | 234.97 | 36.13 | 0.03 | 0.16 | 31540008 | 178623 |
| acctcom | 165 | 1253.99 | 47.06 | 339.34 | 26.64 | 0.29 | 0.14 | 57405662 | 68949 |
| yacc | 58 | 1187.17 | 15.36 | 36.90 | 77.31 | 0.26 | 0.42 | 4096070 | 12093 |
| col | 638 | 1064.40 | 49.01 | 2199.00 | 21.72 | 0.08 | 0.02 | 23835395 | 16903 |
| line | 27184 | 1036.03 | 93.14 | 1941.33 | 11.12 | 0.00 | 0.05 | 925447 | 296142 |
| nroff1.2 | 29 | 909.83 | 17.71 | 56.97 | 51.38 | 0.61 | 0.31 | 11459920 | 18802 |
| delta | 264 | 904.54 | 23.07 | 254.06 | 39.21 | 0.09 | 0.09 | 24219141 | 87164 |
| td | 175 | 886.19 | 25.74 | 159.73 | 34.43 | 0.15 | 0.16 | 1990177 | 15792 |
| ar | 1434 | 872.65 | 61.87 | 309.07 | 14.11 | 0.04 | 0.20 | 189858731 | 428871 |
| m2findD | 144 | 864.29 | 12.54 | 344.13 | 68.94 | 0.09 | 0.04 | 1184947 | 28576 |
| rm | 15319 | 857.97 | 85.65 | 754.20 | 10.02 | 0.01 | 0.11 | 453479 | 433903 |
| acctdusg | 1 | 819.77 | 39.30 | 170.10 | 20.86 | 39.30 | 0.23 | 1812480 | 39744 |
| f77pass1 | 155 | 779.13 | 7.97 | 29.09 | 97.70 | 0.05 | 0.27 | 990027 | 34702 |
| diff | 786 | 767.31 | 32.77 | 260.27 | 23.41 | 0.04 | 0.13 | 22940094 | 97214 |

ATTACHMENT 7.2 (Contd)

Jun 8 04:07 1979 LAST LOGIN Page 1

| | | | | | |
|---|---|---|---|---|---|
| 00-00-00 | dii | 00-00-00 | rudd | 79-06-08 | adm |
| 00-00-00 | absadm | 00-00-00 | s10 | 79-06-08 | agp |
| 00-00-00 | absafr | 00-00-00 | s2 | 79-06-08 | als |
| 00-00-00 | absczs | 00-00-00 | s4 | 79-06-08 | arf |
| 00-00-00 | absjcw | 00-00-00 | s5 | 79-06-08 | cath |
| 00-00-00 | abspvg | 00-00-00 | s6 | 79-06-08 | dal |
| 00-00-00 | absrbm | 00-00-00 | s8 | 79-06-08 | dan |
| 00-00-00 | adm94 | 00-00-00 | s9 | 79-06-08 | denise |
| 00-00-00 | apb | 00-00-00 | scbsa | 79-06-08 | dp |
| 00-00-00 | archive | 00-00-00 | sjm | 79-06-08 | ds52 |
| 00-00-00 | asc | 00-00-00 | srb | 79-06-08 | ech |
| 00-00-00 | badt | 00-00-00 | sys | 79-06-08 | ecp |
| 00-00-00 | btb | 00-00-00 | tgp | 79-06-08 | eric |
| 00-00-00 | bvl | 00-00-00 | tld | 79-06-08 | fld |
| 00-00-00 | bwk | 00-00-00 | ussc | 79-06-08 | fsrepl |
| 00-00-00 | chicken | 00-00-00 | uucpa | 79-06-08 | games |
| 00-00-00 | class | 00-00-00 | uvac | 79-06-08 | graf |
| 00-00-00 | cleary | 00-00-00 | vav | 79-06-08 | herb |
| 00-00-00 | cs | 00-00-00 | wdr | 79-06-08 | hoot |
| 00-00-00 | dbs | 00-00-00 | willa | 79-06-08 | hsm |
| 00-00-00 | deby | 00-00-00 | zooma | 79-06-08 | jac |
| 00-00-00 | dec | 79-06-04 | dws | 79-06-08 | jcw |
| 00-00-00 | demo | 79-06-04 | ewb | 79-06-08 | jel |
| 00-00-00 | dlt | 79-06-04 | kas | 79-06-08 | jfn |
| 00-00-00 | dmr | 79-06-04 | satz | 79-06-08 | kof |
| 00-00-00 | docs | 79-06-04 | uucp | 79-06-08 | krb |
| 00-00-00 | dug | 79-06-05 | bcm | 79-06-08 | leap |
| 00-00-00 | ellie | 79-06-05 | lprem | 79-06-08 | llc |
| 00-00-00 | fsrep2 | 79-06-05 | s7 | 79-06-08 | m2 |
| 00-00-00 | gas | 79-06-05 | sccs | 79-06-08 | m2class |
| 00-00-00 | graphics | 79-06-06 | conv | 79-06-08 | mfp |
| 00-00-00 | hjg | 79-06-06 | dck | 79-06-08 | mhb |
| 00-00-00 | hlb | 79-06-06 | dmt | 79-06-08 | mjr |
| 00-00-00 | inst | 79-06-06 | emp | 79-06-08 | msb |
| 00-00-00 | jfm | 79-06-06 | pah | 79-06-08 | nuucp |
| 00-00-00 | jrh | 79-06-06 | sync | 79-06-08 | paul |
| 00-00-00 | ken | 79-06-06 | tad | 79-06-08 | pdw |
| 00-00-00 | lco | 79-06-07 | ams | 79-06-08 | pris |
| 00-00-00 | learn | 79-06-07 | bin | 79-06-08 | pwbcs |
| 00-00-00 | lppdw | 79-06-07 | dgd | 79-06-08 | pwbst |
| 00-00-00 | lrbb | 79-06-07 | haight | 79-06-08 | rbj |
| 00-00-00 | maj | 79-06-07 | hasp | 79-06-08 | reg |
| 00-00-00 | mar | 79-06-07 | jgw | 79-06-08 | rem |
| 00-00-00 | mash | 79-06-07 | leb | 79-06-08 | rje |
| 00-00-00 | meq | 79-06-07 | ljk | 79-06-08 | rnt |
| 00-00-00 | mifi | 79-06-07 | mep | 79-06-08 | root |
| 00-00-00 | mlc | 79-06-07 | nhg | 79-06-08 | rrr |
| 00-00-00 | mmr | 79-06-07 | nws | 79-06-08 | sl |
| 00-00-00 | mpf | 79-06-07 | qtroff | 79-06-08 | s3 |
| 00-00-00 | plan | 79-06-07 | tbm | 79-06-08 | star |
| 00-00-00 | plum | 79-06-07 | train | 79-06-08 | stock |
| 00-00-00 | pvg | 79-06-07 | whr | 79-06-08 | systst |
| 00-00-00 | rakesh | 79-06-07 | wwe | 79-06-08 | teach |
| 00-00-00 | rfg | 79-06-08 | ? | 79-06-08 | text |
| 00-00-00 | ric | 79-06-08 | abs | 79-06-08 | trc |
| 00-00-00 | rrc | 79-06-08 | absjrk | 79-06-08 | vjf |
| | | | | 79-06-08 | whm |

## ATTACHMENT 7.3

**Files in the /usr/adm directory:**

| | |
|---|---|
| diskdiag | diagnostic output during the execution of disk accounting programs |
| dtmp | output from the **acctdusg** program |
| fee | output from the **chargefee** program, ASCII tacct records |
| pacct | active process accounting file |
| pacct? | process accounting files switched via **turnacct** |
| Spacct?.MMDD | process accounting files for **MMDD** during execution of **runacct** |
| wtmp | active wtmp file for recording connect sessions |

**Files in the /usr/adm/acct/nite directory:**

| | |
|---|---|
| active | used by **runacct** to record progress and print warning and error messages; *active* **MMDD** same as *active* after **runacct** detects an error |
| cms | ASCII total command summary used by **prdaily** |
| ctacct.MMDD | connect accounting records in **tacct.h** format |
| ctmp | output of **acctconl** program, connect session records in **ctmp.h** format |
| daycms | ASCII daily command summary used by **prdaily** |
| dayacct | total accounting records for one day in **tacct.h** format |
| disktacct | disk accounting records in **tacct.h** format, created by **dodisk** procedure |
| fd2log | diagnostic output during execution of runacct (see **cron** entry) |
| lastdate | last day **runacct** executed in **date +%m%d** format |
| lock lock1 | used to control serial use of **runacct** |
| lineuse | tty line usage report used by **prdaily** |
| log | diagnostic output from **acctconl** |
| logMMDD | same as *log* after **runacct** detects an error |
| reboots | contains beginning and ending dates from *wtmp*, and a listing of reboots |
| statefile | used to record current state during execution of **runacct** |
| tmpwtmp | wtmp file corrected by **wtmpfix** |
| wtmperror | place for **wtmpfix** error messages |
| wtmperrorMMDD | same as *wtmperror* after **runacct** detects an error |
| wtmp.MMDD | previous day's wtmp file |

**ATTACHMENT 7.3 (Contd)**

**Files in the /usr/adm/acct/sum directory:**

| | |
|---|---|
| cms | total command summary file for current fiscal in internal summary format |
| cmsprev | command summary file without latest update |
| daycms | command summary file for yesterday in internal summary format |
| loginlog | created by **lastlogin** |
| pacct.MMDD | concatenated version of all pacct files for **MMDD**, removed after reboot by **remove** procedure |
| rprt.MMDD | saved output of **prdaily** program |
| tacct | cumulative total accounting file for current fiscal |
| tacctprev | same as *tacct* without latest update |
| tacct.MMDD | total accounting file for **MMDD** |
| wtmp.MMDD | saved copy of wtmp file for **MMDD**, removed after reboot by **remove** procedure |

**Files in the /usr/adm/acct/fiscal directory:**

| | |
|---|---|
| cms? | total command summary file for fiscal ? in internal summary format |
| fiscrpt? | report similar to **prdaily** for fiscal ? |
| tacct? | total accounting file for fiscal ? |